UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Herramienta para la clasificación y anotación de imágenes de decoloración de hongos

Autor/es

IGNACIO LÓPEZ MENDIVE

Director/es

María Vico Pascual Martínez Losa

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2016-17

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Herramienta para la clasificación y anotación de imágenes de decoloración de hongos

Alumno: Ignacio López Mendive

Tutora:

Vico Pascual Martínez-Losa

Logroño 21, julio, 2017

Agradecimientos

Me gustaría dar las gracias principalmente, a mi tutora Dra. Dña. Vico Pascual Martínez-Losa, profesora del departamento de Matemáticas y Computación de la Universidad de La Rioja, por la paciencia y el esfuerzo que ha dedicado durante la elaboración de este trabajo. A su vez, me gustaría agradecer al Dr. D. Jónathan Heras Vicente por su ayuda incondicional.

Resumen

Actualmente una de las formas que tienen los biólogos de poder medir el crecimiento de un hongo es a través del grado de decoloración que se produce al inocular el hongo en un tinte. El biólogo es el encargado de realizar la asignación predictiva.

El objetivo de este Trabajo de Fin de Grado es realizar el diseño y la implementación de una aplicación web que se conecte a un modelo dado, que permite el reconocimiento predictivo a partir de la imagen del propio hongo.

El patrón de diseño que se seguirá será MVC y el código fuente estará escrito en un lenguaje cuyas características son adecuadas para el desarrollo de aplicaciones web como es Python.

Abstract

At present one of the ways that the biologists can measure the growth of fungus is through the discoloration degree that arises when inoculating the fungus in a dye. The biologist is the responsible for assigning the prediction.

The main purpose of this TFG is to carry out the design and implementation of a web application that directly connects to a given model, which allows the predictive recognition from the fungus image itself.

The design pattern to be followed will be MVC and the source code will be written in a programming language whose properties are suitable for applications web development like Python.

Índice

Capítulo 1: Introducción

1.1. Motivación

Esta memoria recogerá los aspectos más relevantes del Trabajo de Fin de Grado: *HERRAMIENTA PARA LA CLASIFICACIÓN Y ANOTACIÓN DE IMÁGENES DE DECOLORACIÓN DE HONGOS,* realizado por Ignacio López Mendive bajo la tutoría de Vico Pascual, profesora del departamento de Matemáticas y Computación de la Universidad de La Rioja. El tema propuesto por la Universidad de La Rioja no surge como continuación del trabajo realizado durante las prácticas en empresa; además, el alumno ha necesitado aprender unas tecnologías totalmente distintas a las que hasta entonces conocía.

Uno de los aspectos fundamentales de la microbiología es el estudio de la micología, también conocida como estudio de los hongos. Uno de los pilares básicos de esta ciencia se basa en el análisis del crecimiento, es decir, medir el grado de crecimiento de un cierto hongo en un determinado entorno.

Una forma de poder medir el crecimiento en un hongo es la siguiente: en primer lugar se introduce un determinado tinte en unos recipientes cilíndricos denominados discos de Petri (*ver Figura 1*), después se sumerge el hongo, foco de estudio, dentro del compuesto químico. Finalmente, dependiendo del grado de decoloración que surge tras la reacción entre el tinte y el nivel de crecimiento del hongo, el biólogo debe asignar la predicción que considere adecuada para la muestra analizada.

Figura 1: Disco de Petri.

Hay dos formas de realizar el análisis, la primera de ellas es determinar la predicción directamente sobre la muestra y la segunda de ellas es determinar la predicción comparando la reacción obtenida con una muestra de control. Ésta última, está compuesta únicamente por tinte, el mismo que utiliza la muestra con la que se compara, y además jugará un papel muy importante a lo largo del trabajo.

La valoración predictiva del crecimiento realizada por el biólogo, está basada en el rango de símbolos que aparece en la *Tabla 1*, la cual se puede ver a continuación.

Color del tinte	Estado del hongo	Rango de símbolos
Tonalidades muy claras y totalmente transparentes.	Crecimiento muy alto.	+++
Tonalidades amarillentas y parcialmente transparentes.	Crecimiento alto.	++
Mezcla de tonalidades opacas y claras.	Crecimiento moderado.	+
Tonalidades oscuras y opacas.	Crecimiento bajo.	-

Tabla 1: Notación de predicciones y sus características asociadas.

1.2. Antecedentes

El proceso de asignar una predicción a un determinado hongo la realiza el propio biólogo a mano. En la actualidad, ingenieros de la Universidad de la Rioja han desarrollando un software basado en *Machine learning,* capaz de obtener la predicción del crecimiento, a partir de la imagen del propio hongo encerrado dentro de su respectivo disco de Petri.

Al alumno se le ha entregado dicho software definido como un servicio web, con un diseño pobre pero que permite interactuar con el usuario. Actualmente la API de dicho servicio web está disponible en el servidor cuya IP es http://193.146.250.57:8000. El servicio implementa los siguientes métodos de análisis predictivo:

1. **POST: fungi_classification/predictFungiClassUsingOverfeat**

   Este método permite analizar una única imagen de una determinada muestra. Ejemplo de uso:

   curl -X POST -F image=@control.jpg 'http://193.146.250.57:8000/fungi_classification/predictFungiClassUsingOverfeat/'; echo "" ""

   Para éste método, el cliente ha proporcionado una vista. En la *Figura 2* podemos ver el formulario que permite cargar la imagen en el modelo.

Figura 2: Formulario para enlazar una única imagen.

2. **POST: fungi_classification/predictFungiClassUsingOverfeatWithControl**

   Este método recibe dos imágenes: la imagen del hongo de la cual queremos obtener su estimación y su correspondiente imagen de control (*ver Figura 3*). El modelo fusiona ambas imágenes en una, a partir de la cual obtiene la predicción. Hay que tener en cuenta que para este método, el tinte de ambas imágenes debe ser exactamente el mismo.

   A pesar de utilizar dos imágenes, la predicción obtenida se asocia únicamente a la imagen de la muestra. Lo que ocurre, es que al utilizar una imagen de control en el análisis, el modelo tiene mayor información para realizar la predicción, y como

consecuencia, ésta tendrá mayor fiabilidad.

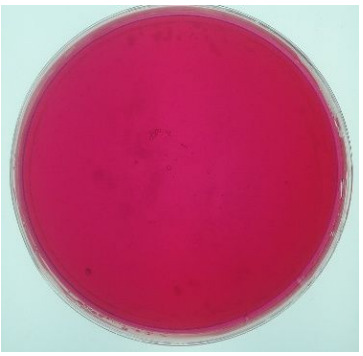


Figura 3: Imagen de control tomada de una muestra de control.

Ejemplo de uso:

curl -X POST -F image=@control.jpg -F imageControl=@control.jpg 'http://193.146.250.57:8000/fungi_classification/predictFungiClassUsingOverfeatWithControl/'; echo "" ""

Es importante destacar que la vista de este método en el modelo proporcionado por el cliente, no está implementada. Por tanto ha sido imposible realizar pruebas con él.

3. **POST: fungi_classification/predictFungiClass/zip/**

   El último método permite cargar un fichero con extensión zip que contenga un conjunto de imágenes de distintos hongos tintados. El modelo se encarga de descomprimir el fichero y analizar una por una, de manera separada, las imágenes contenidas en él. Tanto los hongos como los tintes que aparecen en las imágenes no tienen porqué estar relacionados, puesto que el análisis es individual para cada imagen.

   Ejemplo de uso:

   curl -X POST -F zip=@zip.zip 'http://193.146.250.57:8000/fungi_classification/predictFungiClass/zip/'; echo "" ""

Una vez cargadas las imágenes, se dispone de una vista para utilizarlo que contiene un botón "*Upload and analyse*" (*ver Figura 4*), y que permite cargar las imágenes en el modelo y realizar el análisis correspondiente.

Figura 4: Botón para enviar el formulario al modelo.

La manera en la que el modelo devuelve los datos predictivos depende del método utilizado. Para los que permiten analizar una imagen, utiliza formato JSON, mientras que para el método zip devuelve directamente una página HTML. Los datos que conforman la respuesta son la imagen analizada, aparentemente sin modificar, y la predicción, cuyo valor se corresponde con uno de los mostrados en la *Tabla 1.*

En la *Figura 5* se ve como se muestran los datos tras la predicción, en la vista existente. Como

se ha dicho anteriormente, la interfaz es muy pobre.

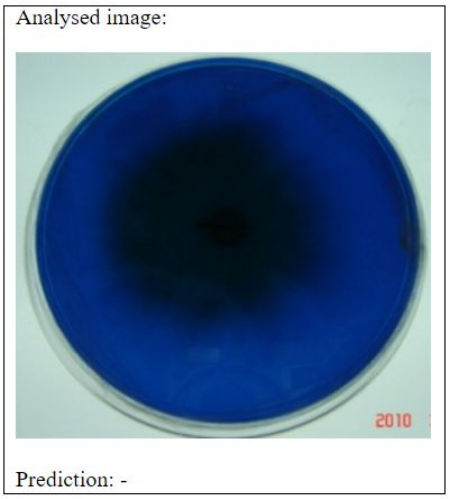


Figura 5: Vista de la predicción realizada por el modelo del análisis de una imágen .

1.3. Objetivos

El trabajo propuesto al alumno consiste en desarrollar una aplicación web completa, que se conecte con el servicio web expuesto anteriormente (*ver apartado 1.2)* y que permita hacer uso de los métodos que éste dispone.

Debido a que el alumno posee un leve conocimiento previo sobre el lenguaje de programación Python, se ha propuesto como objetivo personal desarrollar la aplicación web en dicho lenguaje. Además de que el contexto del proyecto requiere un lenguaje de desarrollo con las características que Python proporciona, el alumno podrá aumentar su conocimiento y destreza en el uso de esta tecnología.

Por otro lado, se ha propuesto como objetivo seguir el patrón de diseño MVC: Modelo-Vista-Controlador. El motivo de esta elección, se debe a que el alumno ha utilizado con anterioridad dicho patrón, pero en un entorno totalmente distinto al que se va a utilizar para desarrollar este proyecto. Por tanto, se ha planteado que el alumno aprenda una manera nueva de implementar el patrón MVC, en otro entorno con unas herramientas diferentes.

El cliente ha proporcionado la parte del modelo al alumno, ésta es la encargada de realizar la estimación predictiva a través del reconocimiento de imágenes. Es importante la total comprensión por parte del alumno del funcionamiento del modelo, para poder desarrollar la vista y el controlador. También se deberá desarrollar la capa de persistencia para poder almacenar los datos en una base de datos.

La parte del controlador será la encargada de acceder al modelo, tanto para enviarle las imágenes a analizar, como para obtener los datos predictivos generados. Finalmente, éste se encargará de enviar dichos datos a la vista, pasándole el control a la misma.

La parte de la vista permitirá, por un lado pasar las imágenes al controlador y, por otro, mostrar los datos obtenidos tras las predicción realizada por el modelo. A su vez, deberá permitir la opción de modificar el resultado obtenido en el caso de que el usuario, experto en

hongos, no esté de acuerdo con él.

Una vez que el usuario esté de acuerdo con los resultados, éstos se podrán almacenar en una base de datos, para su posterior uso. De momento el uso en el que se está pensando es en utilizar las predicciones que se vayan realizando, para reentrenar el modelo. La capa de persistencia será capaz de conectarse directamente con la base de datos.

1.3.1. Requisitos de la aplicación

A continuación, incluimos la lista de requisitos iniciales de la aplicación:

1. La aplicación web permitirá al usuario elegir cuál de las tres operaciones definidas por el modelo quiere realizar:
    1.1. Analizar una imagen.
    1.2. Analizar una imagen disponiendo de una de control.
    1.3. Analizar las imágenes contenidas en un fichero zip.
2. El envío de imágenes al servicio web del cliente se realizará a través de formularios.
3. La aplicación web deberá mostrar al usuario los datos obtenidos.
4. El usuario podrá modificar los datos si no está conforme con la predicción obtenida.
5. El usuario puede decidir guardar los datos obtenidos tras la predicción.
6. La aplicación deberá conectarse con una base de datos para almacenar las predicciones con las imágenes obtenidas.
7. El idioma utilizado en la aplicación web será inglés.
8. La aplicación estará disponible para todos los navegadores.

1.4. Metodología

Antes de realizar la planificación debemos tener en cuenta los siguientes aspectos:

-El tema a desarrollar como TFG, no surge como continuación del trabajo realizado por parte del alumno durante las prácticas curriculares y extracurriculares.

-Es importante destacar el periodo inicial dedicado a la formación y aprendizaje de estas tecnologías, esta estimación dificulta la asignación en el tiempo de las distintas tareas que el alumno debe realizar.

-El alumno realizará el trabajo de manera autónoma, con el seguimiento y control de su tutora de la universidad.

El desarrollo del proyecto se llevará a cabo en su totalidad de Lunes a Domingo durante la segunda mitad del mes de mayo y los meses de junio y julio. También, para la redacción de esta memoria, se empleará un tiempo adicional durante los fines de semana.

En nuestro caso vamos a seguir una metodología ágil. Vamos a planificar un total de 300 horas, divididas en un total de 4 sprints (uno cada dos semanas), todos de 75 horas.

1.4.1. Tareas

A continuación tenemos la lista de tareas, que se deben llevar a cabo para la realización del proyecto, así como el tiempo estimado para la realización de cada una.

1. Tareas iniciales y planificación del proyecto. (15 horas)
2. Estudio de Python [1] bajo el paradigma funcional. (35 horas)
3. Selección del IDE y del framework para la realización de aplicaciones web MVC. Estudio previo y realización de tutoriales con las herramientas seleccionadas. (20 horas)
4. Realización de la parte correspondiente al controlador del MVC. (40 horas)
5. Diseño previo de las vistas del MVC. (10 horas)
6. Implementación de las vistas del MVC utilizando plantillas HTML y CSS. (50 horas)
7. Elección, diseño y creación de la base de datos. (10 horas)
8. Conectar la aplicación MVC desarrollada con la base de datos. (5 horas)
9. Implementación de la capa de persistencia. (15 horas)
10. Realización de pruebas de test previas al cierre del proyecto. (15 horas)
11. Documentación del código realizado. (5 horas)
12. Seguimiento y control. (10 horas)
13. Reuniones. (10 horas)
14. Redacción de la memoria. (40 horas)
15. Preparación de la defensa del TFG. (20 horas)

1.4.2. Sprints

Hemos decidido dividir el trabajo en tres bloques principales, uno para cada tipo de método que ofrece el modelo. En cada bloque, habrá una parte dedicada a la creación del controlador, otra para la creación y diseño de las vistas, y una tercera para la gestión del almacenamiento en la base de datos. Una vez realizadas las partes del controlador y las vistas para el primer método, éstas serán utilizadas como plantillas para el desarrollo de las partes

Vista-Controlador de los dos métodos restantes.

Cabe destacar que en el reparto de tareas por sprint que realizamos a continuación no aparecen aquellas correspondientes a la redacción de la memoria y a las reuniones semanales con el cliente, pero se van a tener en cuenta en cada uno de ellos.

SPRINT 1

- Tareas de iniciación y planificación del proyecto.
- Estudio de Python bajo el paradigma funcional.
- IDE y Framework
    - Analizar y estudiar los posibles IDEs y frameworks, y seleccionar uno.
    - Realizar tutoriales sobre aplicaciones web utilizando el IDE y framework previamente seleccionado.

SPRINT 2

- Creación de la parte del controlador que permita realizar la predicción de una imagen.
- Diseño de las vistas.
- Creación de la estructura básica de las vistas y los formularios.
- Creación de la base de datos.
- Construcción de la primera parte de la capa de persistencia correspondiente al método predictivo de una imagen.

SPRINT 3

- Creación de la parte del controlador que permita realizar la predicción de una imagen utilizando su imagen de control asociada.
- Construcción de la parte de la capa de persistencia correspondiente al método predictivo de una imagen utilizando su correspondiente imagen de control.
- Diseñar e implementar las interfaces. (Parte 1)

SPRINT 4

- Creación de la parte del controlador que permita realizar la predicción de las imágenes contenidas en un archivo zip.
- Construcción de la parte de la capa de persistencia correspondiente al método predictivo de un zip de imágenes.
- Diseñar e implementar las interfaces. (Parte 2)
- Pruebas de test.

Nota: A partir de ahora los métodos del modelo presentados anteriormente, aparecerán referenciados a lo largo de la memoria de la siguiente manera:

- Primer método: correspondiente con el método que permite analizar una imagen.
- Segundo método: correspondiente con el método que permite analizar una imagen utilizando su correspondiente imagen de control.
- Tercer método: correspondiente con el método que permite analizar un fichero zip de imágenes.

Capítulo 2: Sprint 1

2.1. Sprint planning

Las tareas previstas para el primer sprint son las siguientes:

-Tareas de iniciación y planificación del proyecto. Esta tarea ya ha sido realizada y documentada en el *Capítulo 1*.

-Estudio de Python bajo el paradigma funcional.

-IDE y Framework:

- Analizar los posibles IDEs y frameworks, y seleccionar uno.
- Realizar tutoriales sobre aplicaciones web utilizando el IDE y framework previamente seleccionado.

2.2. Entorno de desarrollo y framework

En primer lugar, el alumno ha realizado una búsqueda informativa de los posibles IDEs y frameworks, con el fin de seleccionar aquellos que nos permitan trabajar y desarrollar nuestra aplicación web en Python.

Se han evaluado los siguientes IDEs:

- Pycharm IDE. De entre sus características más relevantes destaca el soporte de frameworks para desarrollo web, como Django, Pyramid y Web2py, entre otros. También cabe destacar que es multiplataforma.
- PyDev IDE. Es un plugin de código abierto que se puede instalar en Eclipse. Permite la integración del framework Django, y entre sus características destacan el autocompletado y análisis de código.
- Wing IDE. Su característica más potente es el debugging, ya que permite la depuración de procesos múltiples y subprocesos de manera remota.

De entre los distintos frameworks analizados para el desarrollo web en Python, destacan los siguientes:

- Django [2], es el mayor framework de desarrollo web de alto nivel existente para Python. Este permite el desarrollo rápido y el diseño limpio y pragmático. Además posee una gran comunidad, lo que favorece la documentación y apoyo en gran escala cuando se está programando.
- Pyramid, es un framework pequeño, rápido y flexible. Posee una gran documentación, perfecto para principiantes en el mundo de la programación web en Python.
- Flask, es un micro-framework simple y ligero, constituido por un conjunto de módulos.

No proporciona una gran funcionalidad, pero permite añadir una serie de extensiones.

Tras realizar una valoración exhaustiva de los distintos IDEs y frameworks presentados, se ha decidido utilizar como IDE, PyCharm, y como framework Django.

En primer lugar se ha elegido PyCharm, debido a que es una aplicación completa y robusta. Además es muy sencilla e intuitiva, sobre todo para aquellas personas que no tienen un gran conocimiento sobre desarrollo web en Python, como es el caso del alumno. Las otras dos aplicaciones restantes han sido descartadas ya que se ha considerado que pueden aportar una funcionalidad más reducida que PyCharm.

En segundo lugar se pensó en utilizar Pyramid como framework, ya que sus características parecían adecuadas, pero tras consultar con un experto de programación web en Python, éste aconsejó el uso de Django, debido a que dispone de un ORM (denominado *Querysets*) que facilitará la construcción de la capa de persistencia, permitiendo realizar consultas SQL a bases de datos sin utilizar *querys*. Además, Django es un framework de alto nivel escrito en Python, por lo que va a facilitar al alumno entender y desarrollar el código de manera rápida, así como realizar una programación que favorezca la reutilización del código.

2.3. MVC en Django

Tras instalar los programas seleccionados y haber configurado el entorno de desarrollo, se ha podido comenzar con la creación de la aplicación. Es importante destacar que la creación de un proyecto en Django no es trivial, puesto que requiere de varias acciones de configuración, así como comprender con exactitud cada uno de los scripts que se generan y qué información incluye cada uno de ellos. Para una información más detallada *ver Anexo I.*

En la *Figura 6* se observa la estructura del directorio que compone nuestra aplicación

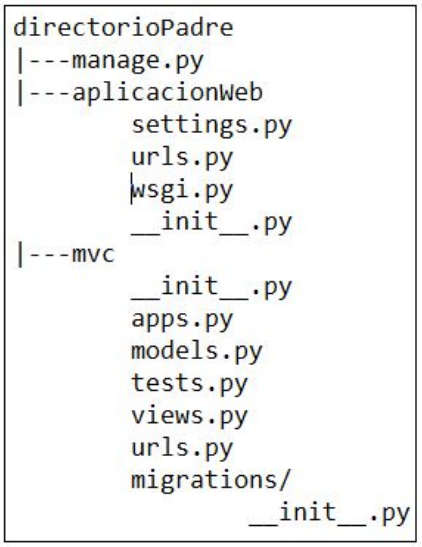


Figura 6: Estructura del proyecto en Django.

Para entender la implementación del MVC que vamos a desarrollar, es importante definir dónde va ir implementada cada una de las partes dentro del proyecto Django.

- El Modelo, aunque en nuestro caso ya viene dado, suele implementarse en el fichero

models.py. Éste es el lugar donde se definen los objetos que se guardarán en la base de datos.

- El Controlador se implementará en el fichero *views.py,* aquí se definirán las funciones que permitirán controlar las peticiones que se realicen y consecuentemente el despliegue de las vistas elaboradas.
- La Vista, serán plantillas (templates) HTML y hojas de estilo CSS. Tienen el propósito de determinar qué datos serán visualizados y la validación de los mismos, a través de formularios y JavaScript.

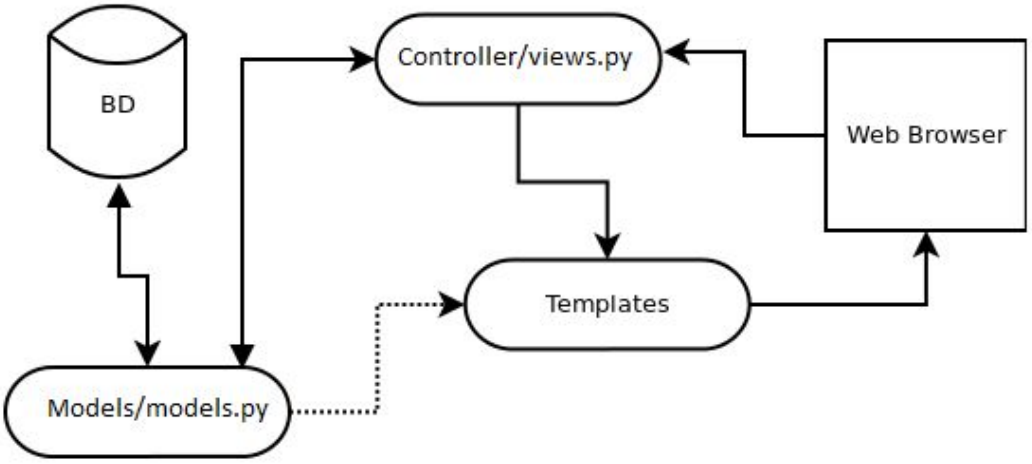


Figura 7: Patrón MVC en Django.

2.4. Sprint retrospective

Todas las tareas planificadas han sido realizadas, por tanto no queda nada pendiente para el siguiente sprint. Por el momento, la planificación se está respetando tal y como se decidió en un principio. El número de horas dedicadas al primer sprint han sido 60.

Capítulo 3: Sprint 2

3.1. Sprint planning

Para el segundo sprint, el alumno va a desarrollar la parte del controlador, la parte de las vistas y la parte de la persistencia correspondiente al primer método ofrecido por el modelo: “Analizar una imagen”.

Las tareas previstas a lo largo del sprint son las siguientes:

-Creación de la parte del controlador que permita realizar la predicción de una imagen.

-Creación de la estructura básica de los formularios sobre los que se basan las vistas.

-Creación de la base de datos.

-Construcción de la parte de la capa de persistencia correspondiente al método predictivo de una imagen.

-Diseño de las vistas.

3.2. Desarrollo

3.2.1. Menú inicial

El primer paso es definir una página de inicio. Aprovechando que nuestro modelo realiza tres métodos diferentes, se ha decidido que la primera vista sea un menú con tres enlaces, uno para cada acción, como aparece en la *Figura 8*.

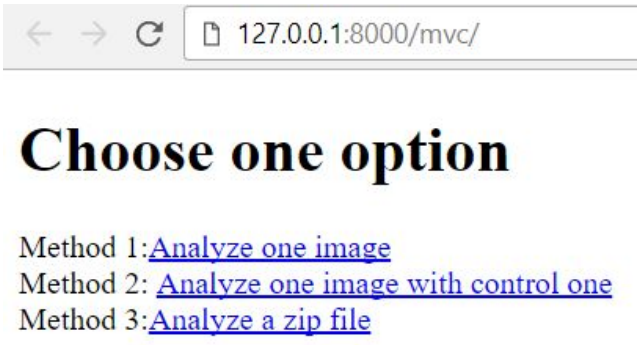


Figura 8: Estructura del menú principal de la aplicación.

En la parte del controlador se ha definido una función inicial que permite cargar la vista *Menu.html*. Esta función, como todas las demás que se desarrollarán en el fichero *views.py*, devolverá un método *render*() (*ver Figura 9*). Este método tiene como parámetros de entrada: la petición y la vista a mostrar, su principal función es renderizar[1] el template. Este método permite almacenar los parámetros en el *request*, para poder usarlos en las vistas.

[1] renderizar: Construir la vista a partir del modelo.

```
def index(request):
    template = loader.get_template("Menu.html")
    return render(request, 'Menu.html')
```

Figura 9: Función inicial del controlador asociada a la vista del menú.

La vista del menú permitirá a los usuarios de la aplicación acceder a las diferentes funcionalidades que ofrece el modelo.

3.2.2. El controlador para el análisis de una imagen

Antes de implementar el controlador, es necesario crear un fichero llamado *forms.py* donde poder definir los formularios que vamos a utilizar en las vistas. Para crear un formulario simplemente basta con crear una clase cuyos atributos se corresponderán con los distintos inputs que contendrá dicho formulario. En la *Figura 10* podemos ver cómo se ha definido el formulario para el primer método del modelo.

```
class Formulario(forms.Form):
    image=forms.ImageField()
```

Figura 10: Definición de un formulario en el fichero forms.py con un campo imagen.

La estructura HTML de los formularios que usaremos para cargar las imágenes en el modelo, se basarán en la que aparece en la *Figura 11*.

```
<form id="Formulario" name="f1" action="" method="POST" enctype="multipart/form-data">
    {% csrf_token %}
    {{form.as_p}}
    <input type="submit" value="Enviar datos">
    <br>
    <a href="/mvc/">Volver al menú.</a>
</form>
```

Figura 11: Estructura del formulario que permite cargar imágenes en el modelo.

Hay que tener en cuenta ciertos aspectos a la hora de definir los formularios de subida de imágenes. En primer lugar, es necesario que el atributo *id* del formulario en el HTML y el nombre de la clase del fichero *forms.py* coincidan. Como vamos a enviar una imagen mediante un formulario, el método HTTP que se va a usar es POST y el valor del atributo *enctype* va a ser *multipart/form-data*. Por otro lado, el comando *{{form.as_p}}* nos permite mostrar los atributos que hemos definido en la clase "Formulario" dentro del fichero *forms.py*, transformados en inputs de HTML. Además definiendo el formulario de esta manera, se realiza automáticamente la validación de sus campos antes de ser enviado. Por último, el método *{%csrf_token%}* es un mecanismo de seguridad que previene a la aplicación ante el envío de código malicioso por parte de usuarios malintencionados.

Nota: La manera de escribir código Python embebido en HTML se realiza mediante la notación {{...}} o {%...%}.

Una vez configurado lo anterior, se va a realizar la parte correspondiente al controlador de los distintos métodos que ofrece el modelo. Para cada método se definirán dos funciones. La

primera de ellas se va a encargar de contactar con el modelo enviando las imágenes del formulario a través de un método POST, además se ocupará de obtener la respuesta en formato JSON y añadir a la sesión aquellos datos que interese mostrar en la vista correspondiente a la predicción. Y la segunda, se encargará de obtener los datos de la sesión y pasarlos a la vista, además en esta se definirá la persistencia de la aplicación.

Para la realización de la primera parte del controlador, correspondiente al método de realizar la predicción de una imagen, el alumno ha partido del siguiente comando que el cliente le proporcionó durante la presentación del modelo:

> curl -X POST -F image=@control.jpg 'http://193.146.250.57:8000/fungi_classification/predictFungiClassUsingOverfeat/'; echo "" "".

Podemos ver a continuación, como se ha traducido el comando en Python:

```
url = 'http://193.146.250.57:8000/fungi_classification/predictFungiClassUsingOverfeat/'
files = {'image': open('C:/Users/ignac/Downloads/blog/Imagenes/' + fileName, 'rb')}
response = requests.post(url, files=files).text
data = json.loads(response)
```

Por el momento se han utilizado URLs absolutas, futuramente se configurarán rutas relativas.

Ha sido necesario previamente, instalar la librería *requests*, la cual permite elaborar la respuesta de las distintas peticiones HTTP [3].

El objeto JSON de la respuesta obtenida tiene tres atributos (*ver Figura 12)*, la URL de la imagen relativa al modelo del cliente, la predicción y un booleano, este último tendrá valor *True* cuando la petición se ha realizado con éxito. El formato JSON facilita la obtención de los parámetros de la respuesta, y a su vez hace más sencillo el intercambio de parámetros entre el controlador y la vista.

Figura 12: objeto JSON devuelto por el modelo tras analizar una imagen.

En la vista de destino, se mostrará la imagen analizada y la predicción realizada por el modelo. Queda pendiente establecer con el cliente qué datos adicionales se desea mostrar y cómo.

3.2.3. Selección, creación y diseño de la base de datos

Una vez obtenidas las predicciones del modelo, el siguiente paso es almacenar los datos obtenidos en formato JSON en una base de datos. Para ello el alumno ha tenido que realizar con anterioridad, un estudio de las posibles bases de datos compatibles con Django y sus características principales, para así elegir aquella con la que trabajar. De entre las bases de datos disponibles se han estudiado las siguientes:

- MySQL, es una base de datos multiplataforma con multitud de documentación en la red, lo que facilita el uso y el aprendizaje de la misma al usuario. Además es software libre y es una de las más utilizadas en aplicaciones web.
- SQLite, es una herramienta de software libre creada para ser utilizada principalmente en dispositivos móviles. Es un tipo de base de datos embebida pensada para

aplicaciones pequeñas. Sin embargo no es escalable, no tiene usuarios ni permisos y además no favorece al rendimiento de las aplicaciones que lo usan.
- Oracle, es el motor de base de datos relacional más usado a nivel mundial, accesible para todo tipo de plataformas. Sin embargo es un servicio de pago.
- Microsoft SQL Server, es un sistema de gestión de bases de datos muy útil para manejar y obtener datos de Internet. Sin embargo, su uso es muy restrictivo y la relación calidad precio está muy por debajo comparada, por ejemplo, con Oracle.

Tras el análisis presentado, se ha optado por utilizar MySQL debido a que es fácil de usar y fácil de aprender, permitiendo al usuario gestionar la base de datos de una manera muy intuitiva. Además su instalación y configuración es muy sencilla [5], lo que favorecerá al alumno a la hora de comenzar con la creación de la capa de persistencia de la aplicación web.

3.2.4. La capa de persistencia para el análisis de una imagen

A continuación, se va a construir la parte de la capa de persistencia de la aplicación correspondiente al método del modelo: "Análisis de una imagen". La implementación de esta primera parte servirá al alumno como guía para la creación de las partes restantes de la capa de persistencia. En primer lugar se ha creado la base de datos, la cual se ha decidido llamar *fungi.* Tras instalar los paquetes referentes a los conectores de MySQL para Python, debemos modificar en el fichero de configuración de nuestro proyecto *settings.py*, el apartado relativo a las bases de datos (*ver Figura 13*).

```
DATABASES = {
    'default': {
        'ENGINE':'django.db.backends.mysql',
        'NAME':'fungi',
        'USER':'nacho',
        'PASSWORD':'adminadmin'

    }
}
```

Figura 13: Configuración de la BD MySQL en el fichero settings.py.

Es muy importante que tras este proceso, se realice la migración de los cambios producidos a través de la consola de comandos (*ver Anexo I*). Esto permitirá empezar a crear las tablas dentro de la base de datos que hemos creado.

La capa de persistencia de nuestra aplicación va a ser muy sencilla, solo se va a encargar de insertar en la base de datos MySQL los datos obtenidos tras la predicción realizada por el modelo del cliente. Para poder llevar a cabo el almacenamiento de los datos, el alumno ha tenido que desarrollar en el modelo de la aplicación, que hasta ahora estaba vacío, una clase denominada *ImagenyPrediccion*. Los atributos de esta clase se corresponden con los tipos de datos que vamos a almacenar, en la base de datos, para cada predicción.

- *id*, de tipo entero. Se corresponde con la clave primaria. Es un campo que funciona a modo de contador, éste se va a ir autogenerando cada vez que se realiza una inserción en la base de datos.
- *nombreImagen*, de tipo String. Se corresponde con las rutas relativas de las imágenes respecto a la aplicación.
- *prediccion*, de tipo String. Se corresponde con los valores predictivos realizados por el

modelo de una imagen analizada (*ver Tabla 1*).

Los atributos *nombreImagen* y *predicción* son requeridos, por tanto nunca van a ser nulos ni vacíos. Es importante recordar que los atributos que el cliente quiera almacenar de cada imagen analizada en la base de datos pueden variar, ya que no los ha especificado todavía.

Es obligatorio que la tabla creada (*ver Figura 14*) haga referencia a la clase de nuestro modelo llamándose *mvc_imagenyprediccion*, es decir *directorioRaizAplicacion_nombreClaseModelo*. Además, el nombre y el tipo de los campos de la tabla deben coincidir exactamente con los atributos de la clase definida en nuestro modelo (*ver Figura 15*).

```
mysql> CREATE TABLE mvc_imagenyprediccion (
        ID INT AUTO_INCREMENT,
        nombreImagen VARCHAR(100) NOT NULL,
        prediccion VARCHAR(4) NOT NULL,
        PRIMARY KEY(ID));
```

Figura 14: Creación de la tabla con el nombre y los campos correspondientes a la clase de nuestro modelo.

```
class ImagenyPrediccion(models.Model):
    id = models.AutoField(primary_key=True);
    nombreImagen=models.CharField(max_length=100)
    prediccion = models.CharField(max_length=4)
```

Figura 15: Clase ImagenyPredicción definida en el fichero models.py.

Cabe recordar que Django proporciona un ORM para acceder a las bases de datos sin necesidad de realizar consultas. Por tanto, para insertar una fila en la base de datos, lo primero que hay que hacer es crear una instancia de la clase definida en nuestro modelo usando argumentos por nombre, es decir asignando el nombre de los atributos del modelo a los elementos imagen y predicción obtenidos en la predicción. Este primer paso de instanciar un objeto no modifica la base de datos. Para guardar el registro en la base de datos bastaría con llamar al método *save()* del objeto creado. Este método juega un doble papel, puede funcionar como un INSERT de una consulta SQL, en el caso de que sea llamado por primera vez por el objeto, o un UPDATE, cuando no es la primera vez que el objeto llama a dicho método y además se ha realizado alguna modificación en alguno de sus atributos.

Finalmente en la *Figura 16* se puede observar la tabla creada con algunos ejemplos almacenados en ella.

```
mysql> select * from mvc_imagenyprediccion;
+----+-----------------------------------------------------+------------+
| ID | nombreImagen                                        | prediccion |
+----+-----------------------------------------------------+------------+
|  1 | /media/fungi-classification/2017-06-15-15-38-06.jpg | +++        |
|  2 | /media/fungi-classification/2017-06-15-15-38-32.jpg | +++        |
|  3 | /media/fungi-classification/2017-06-15-15-38-47.jpg | ++         |
|  4 | /media/fungi-classification/2017-06-15-15-39-07.jpg | -          |
+----+-----------------------------------------------------+------------+
```

Figura 16: Tabla mvc_imagenyprediccion y sus correspondientes columnas.

3.3. Sprint retrospective

Se ha logrado realizar las siguientes tareas:

-Creación de la parte del controlador que nos permita realizar la predicción de una imagen.

-Creación de la estructura básica de las vistas y los formularios.

-Creación de la base de datos.

-Construcción de la parte de la capa de persistencia correspondiente al método predictivo de una imagen.

Para su realización se han invertido un total de 65 horas.

Queda pendiente para el siguiente Sprint la tarea:

-Diseño de las interfaces.

Capítulo 4: Sprint 3

4.1. Sprint planning

Durante este sprint el alumno se va a encargar de implementar la parte del controlador, la parte de las vistas y la parte de la persistencia correspondiente al segundo método que ofrece el modelo: “Analizar una imagen acompañada de otra de control”.

Las tareas previstas a lo largo del sprint son las siguientes:

-Diseño de las interfaces. Esta tarea quedó pendiente de realizarse en el anterior sprint. Se va a posponerla al final del sprint para continuar con el controlador y la persistencia.

-Creación de la parte del controlador que permita realizar la predicción de una imagen utilizando su imagen de control asociada.

-Construcción de la parte de la capa de persistencia correspondiente al método predictivo de una imagen utilizando su correspondiente imagen de control.

-Primera parte de la implementación del diseño de las interfaces.

4.2. Desarrollo

4.2.1. El controlador para el análisis de una imágen con su imagen de control

Para el desarrollo de esta parte el alumno ha seguido la misma dinámica que utilizó en la parte del controlador del primer método. Para ello, ha traducido el método dado por el cliente:

curl -X POST -F image=@control.jpg -F imageControl=@control.jpg 'http://193.146.250.57:8000/fungi_classification/predictFungiClassUsingOverfeatWithControl/'; echo "" "",

de la siguiente manera:

```
url = 'http://193.146.250.57:8000/fungi_classification/predictFungiClassUsingOverfeatWithControl/'
fileName = request.FILES['image'].name
fileNameControl = request.FILES['control'].name
filePath = os.path.realpath("../blog/Imagenes/" + fileName)
filePathCon = os.path.realpath("../blog/Imagenes/" + fileNameControl)

files = {'image': open(filePath, 'rb'),
         'imageControl': open(filePathCon, 'rb')}

response = requests.post(url, files=files).text
data = json.loads(response)
```

Es muy similar a la traducción del primer método del modelo. Las únicas diferencias son que la URL de la petición es distinta y que se ha añadido un nuevo parámetro correspondiente a la imagen de control, respecto a la cual el método realiza la petición. En el caso de fallo al enviar

la imagen de control, aparece el siguiente mensaje de error en formato JSON:

```
{'success': False, 'Error': 'You must upload two images'}
```

Por otro lado, una vez recibido el JSON de la respuesta, como en el que aparece en la *Figura 17*, el alumno ha percibido que éste es incompleto. Al parecer la implementación del modelo del cliente para este método está sin acabar. Se observa además, que el JSON obtenido es distinto al del método uno *(ver Figura 12)*, ya que solo contiene la predicción y el booleano, no hay rastro de la URL relativa de la imagen analizada.

```
{'prediction': '+', 'success': True}
```

Figura 17: JSON obtenido como respuesta del segundo método.

Tras esta cuestión, se debe realizar una reunión con el cliente para aclarar varios aspectos referentes a la parte del modelo del segundo método dado, y ver si es necesario que el alumno haga cambios en él. Como durante la primera semana de este sprint el cliente va a estar ausente, se ha decidido posponer la realización de las tareas restantes previstas para este sprint, para el siguiente sprint y adelantar otras.

Así, la nueva lista de tareas para el sprint pasa a ser la siguiente:

-Creación de la parte del controlador que permita realizar la predicción de unas imágenes contenidas en un fichero zip.

-Construcción de la parte de la capa de persistencia que permita almacenar las predicciones de un archivo zip de imágenes.

-Diseño de las interfaces. Esta tarea se mantiene del sprint planning anterior.

-Primera parte de la implementación del diseño de las vistas.

4.2.2. El controlador para el análisis de un fichero zip de imágenes

Para la realización de la parte del controlador correspondiente al tercer método que ofrece el modelo, el alumno ha seguido exactamente el mismo procedimiento que en los dos anteriores. La traducción del siguiente método dado por el cliente,

curl -X POST -F zip=@zip.zip
'http://193.146.250.57:8000/fungi_classification/predictFungiClass/zip/';
echo "" "",

es de la forma:

```
url = 'http://193.146.250.57:8000/fungi_classification/predictFungiClass/zip/'
files = {'zip': open(filePath, 'rb')}
response = requests.post(url, files=files).text
```

La única diferencia con la implementación de los dos métodos anteriores, es la URL de la petición y el parámetro de la petición POST, que es la ruta de un archivo de imágenes con extensión zip. Ahora bien, al obtener la respuesta, el alumno se ha dado cuenta de que el código devuelto por el modelo, correspondiente con las predicciones de las imágenes

contenidas en el zip, no está en formato JSON, sino que es una página HTML completa. Ésto dificulta al alumno poder obtener los datos de la respuesta para, posteriormente, mostrarlos en la vista de destino.

Es por ello que se debe obtener fragmentos del código HTML de alguna manera. En un principio se pensó en recorrer el HTML obtenido mediante JQuery a través del árbol DOM, pero finalmente se descartó la idea debido a que el aprendizaje de una nueva tecnología supondría un consumo de horas excesivo. Tras investigar en la web, se decidió que la manera más eficaz de obtener HTML es realizando un parseo directo de la respuesta obtenida. Por tanto, surge una nueva tarea para realizar durante el sprint:

> -Parseo del documento HTML obtenido como respuesta tras el análisis del archivo zip de imágenes.

Nota: los datos que se quieren obtener de la respuesta HTML son las predicciones y las rutas relativas de las imágenes.

Parseo de la respuesta HTML dada por el modelo

El alumno ha necesitado invertir un tiempo previo a la realización del *scraping*[2], para buscar información acerca de las distintas maneras de realizar el parseo de un texto HTML en Python. Se han estudiado dos maneras de realizarlo:

> -Crear una clase *HTMLParser* en el modelo que use la librería *HTMLParser*.
>
> -Utilizar la librería *BeautifulSoup* directamente sobre el controlador.

Se ha decidido descartar crear la clase *HTMLParser*, debido a que el código es tedioso de implementar y ocupa gran cantidad de espacio, afectando al rendimiento de nuestra aplicación. Por ello, se ha decidido utilizar la librería *BeautifulSoup* de Python [6]. Ésta proporciona una gran cantidad de métodos que facilitan, tanto el acceso directo a los elementos del documento HTML, como la obtención de fragmentos de código del mismo.

Con el método de *BeautifulSoup*, *find_all("tag"),* se han obtenido fragmentos de la respuesta HTML, que mediante distintas manipulaciones con los Strings, el alumno ha sido capaz de obtener dos listas, una de ellas correspondiente a las URLs de las imágenes y la otra a las predicciones de cada imagen. El siguiente paso ha sido enviar a la vista de destino, estos parámetros obtenidos tras el parseo. Para ello, a partir de las dos listas obtenidas anteriormente, se ha creado una lista de objetos del tipo definido en nuestro modelo: *ImagenyPrediccion*. El problema surge cuando se ha añadido la lista de objetos a la sesión para poder recuperarla en la vista, ya que una lista de objetos no es JSON serializable.

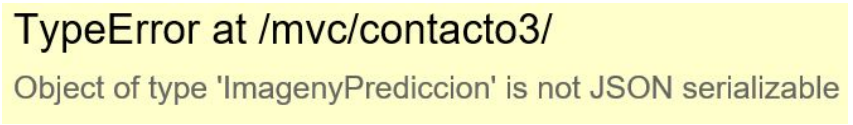


Figura 18: Excepción JSON no serializable.

Para solventar el problema, el alumno ha realizado el parseo a JSON de los distintos objetos contenidos en la lista. Para poder llevarlo a cabo, Python permite usar una estructura de

[2] Web Scraping o Scraping: conjunto de técnicas que se utilizan para obtener de forma automática el contenido que hay en páginas web a través de su código HTML.

datos denominada diccionario. Mediante la función *dict()*, se puede dotar a los objetos contenidos en la lista, de un formato similar a JSON, permitiendo en este caso, serializar por completo la lista de objetos para ser enviada a la vista de destino.

4.2.3. La capa de persistencia para el análisis de un fichero zip de imágenes

Para la capa de persistencia hemos realizado el mismo mecanismo que en el anterior sprint. La diferencia es que tenemos que construir una lista de objetos de tipo *ImagenyPrediccion* para guardarlos en la base de datos, a partir de dos listas de Strings, una que contiene las rutas relativas de las imágenes realizadas por el modelo del cliente y otra que contiene las predicciones. Como ambas listas tienen la misma longitud, Python permite recorrerlas de manera simultánea utilizando el método *zip(lista1, lista2).* Finalmente, llamaremos al método *save()* para insertar las filas en la base de datos.

Nota: Dado un archivo zip analizado, el usuario tiene dos opciones a la hora de almacenar los datos, o se almacenan todas las predicciones de todas las imágenes contenidas en el zip o se descartan todas las predicciones.

4.2.4. Diseño de las interfaces

Para el diseño de las vistas se ha decidido utilizar una aplicación de prototipado de interfaces llamada *MockupScreen* [7]. Esta herramienta ha permitido realizar en tiempo real, el diseño de las interfaces de la aplicación web, además ofrece una opción que permite interactuar directamente con las vistas al igual que si se estuviera usando la aplicación que se quiere desarrollar.

A continuación se pueden ver los diseños, realizados por el alumno, de las distintas vistas de la aplicación web. Éstos están pendientes de ser evaluados por el cliente, por lo que pueden no corresponderse con los prototipos finales de la aplicación.

En primer lugar se presenta la página inicial de la aplicación a modo de menú, a través del cual se puede acceder a las tres posibles funcionalidades predictivas que el modelo ofrece.

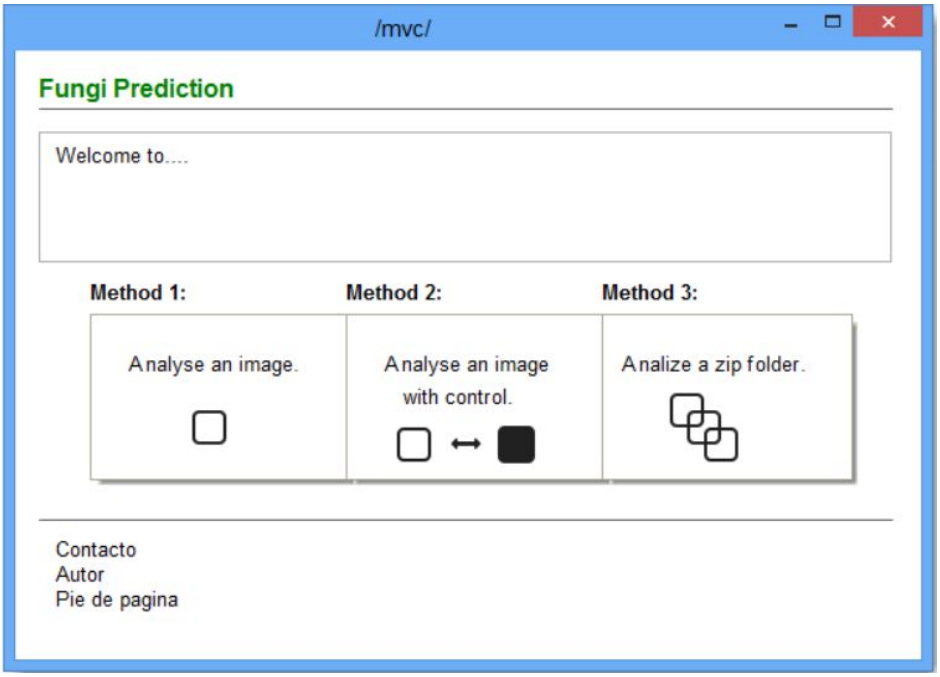


Figura 19: Prototipo de la vista menú principal.

Dependiendo de la opción elegida, aparece un formulario para enlazar: una imagen, dos imágenes (una de ellas de control) o un fichero zip de imágenes respectivamente, como se observa continuación. Estas vistas ofrecen la posibilidad de volver al menú inicial mediante un enlace.

Prototipo del formulario para el método que permite analizar una imagen.

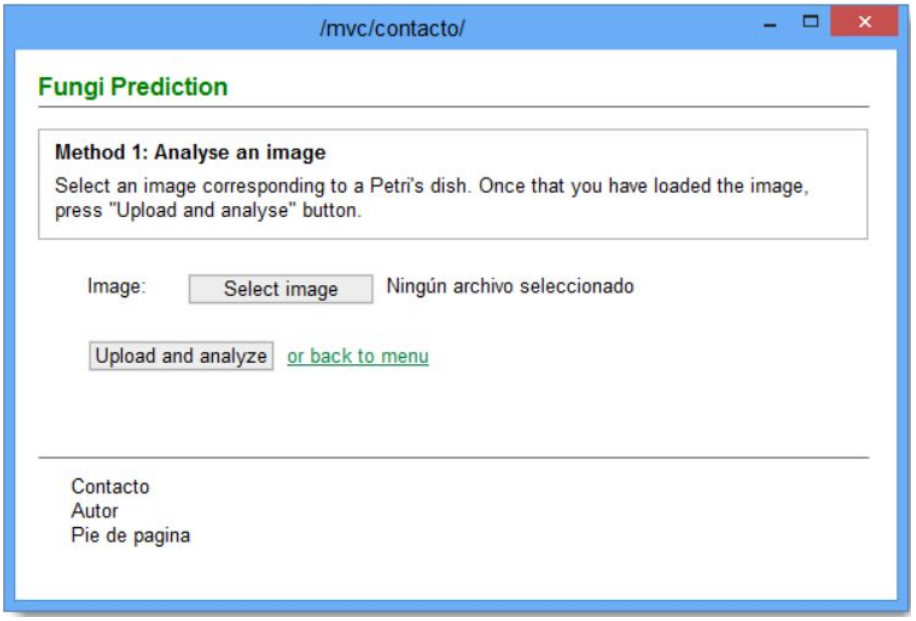


Figura 20: Prototipo de la vista formulario de enlace para el análisis de una imagen.

Prototipo del formulario para el método que permite analizar una imagen utilizando su imagen de control asociada.

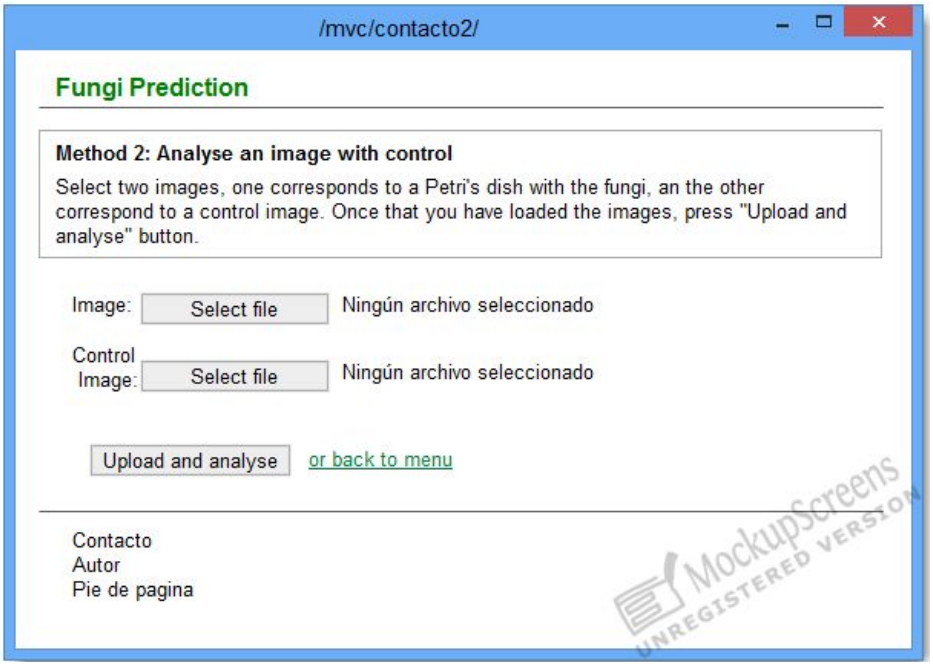


Figura 21: Prototipo de la vista formulario de enlace para analizar una imagen utilizando su correspondiente imagen de control.

Prototipo del formulario para el método que permite analizar un conjunto de imágenes dentro de un fichero zip.

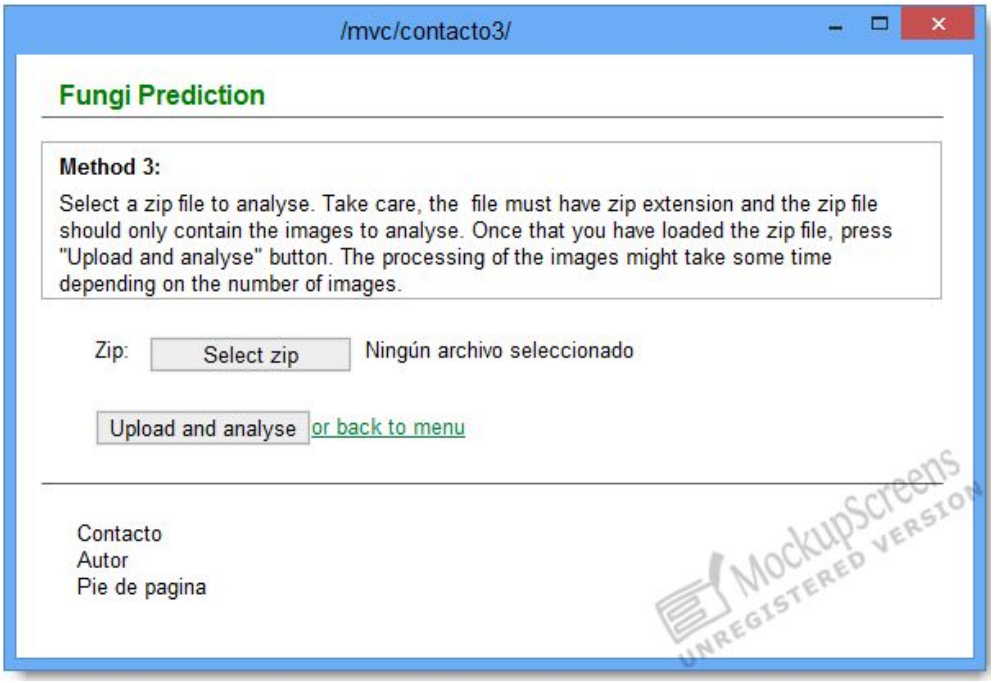


Figura 22: Prototipo de la vista formulario para enlazar un fichero zip de imágenes.

Una vez seleccionado el fichero a analizar se pulsa el botón "*Upload and analyze*", acto seguido se accede a la vista donde se mostrará la predicción realizada por el modelo. Se ha pensado en

mostrar la imagen o imágenes analizadas, y la predicción o predicciones obtenidas. El campo predicción se ha decidido introducirlo dentro de un *textbox* para que pueda ser modificado antes de ser almacenado en la base de datos, en dicha caja de texto se realizarán las comprobaciones necesarias para que los posibles datos a introducir sean exclusivamente aquellos que aparecen en la *Tabla 1*.

A continuación se observan las vistas que alojan las predicciones obtenidas realizadas por el modelo, para los tres métodos presentados.

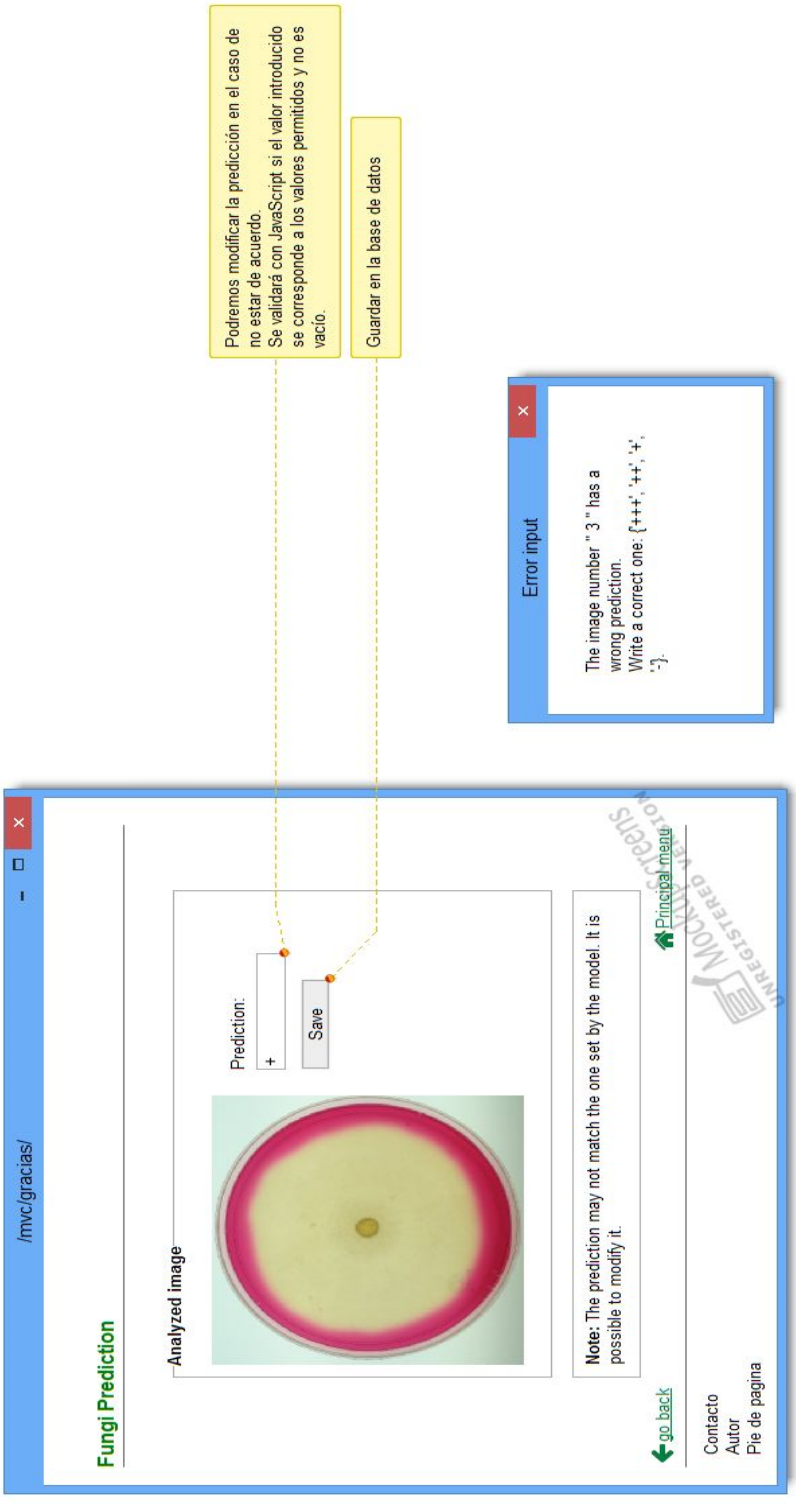


Figura 23: Prototipo de la vista predictiva tras el análisis de una imagen.

Para la vista predictiva del método correspondiente al análisis de una imagen con la de control, se ha pensado mostrar la imagen de control, con el fin de informar visualmente al usuario sobre el tinte utilizado.

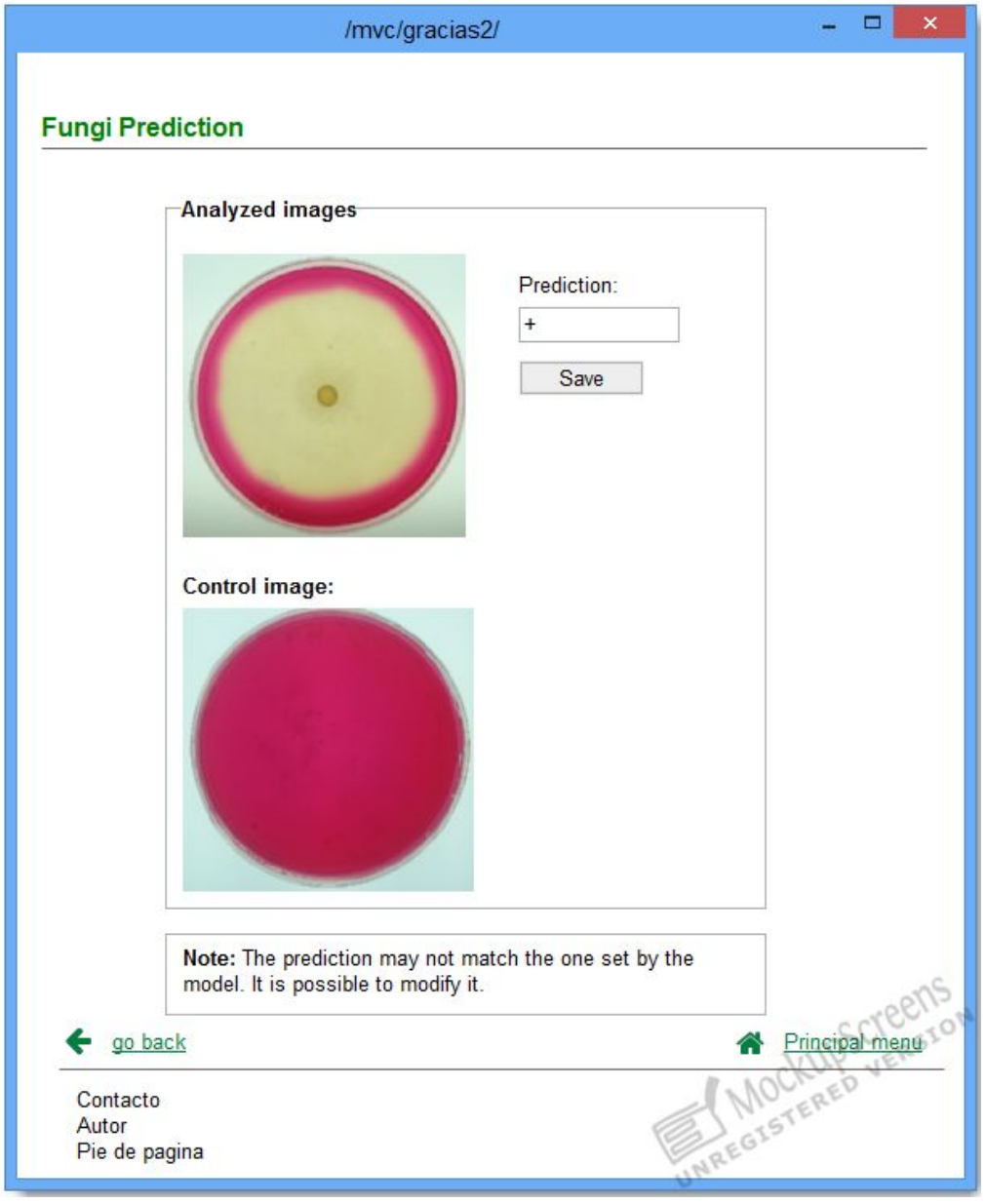


Figura 24: Prototipo de la vista predictiva tras el análisis de una imagen con su correspondiente imagen de control asociada.

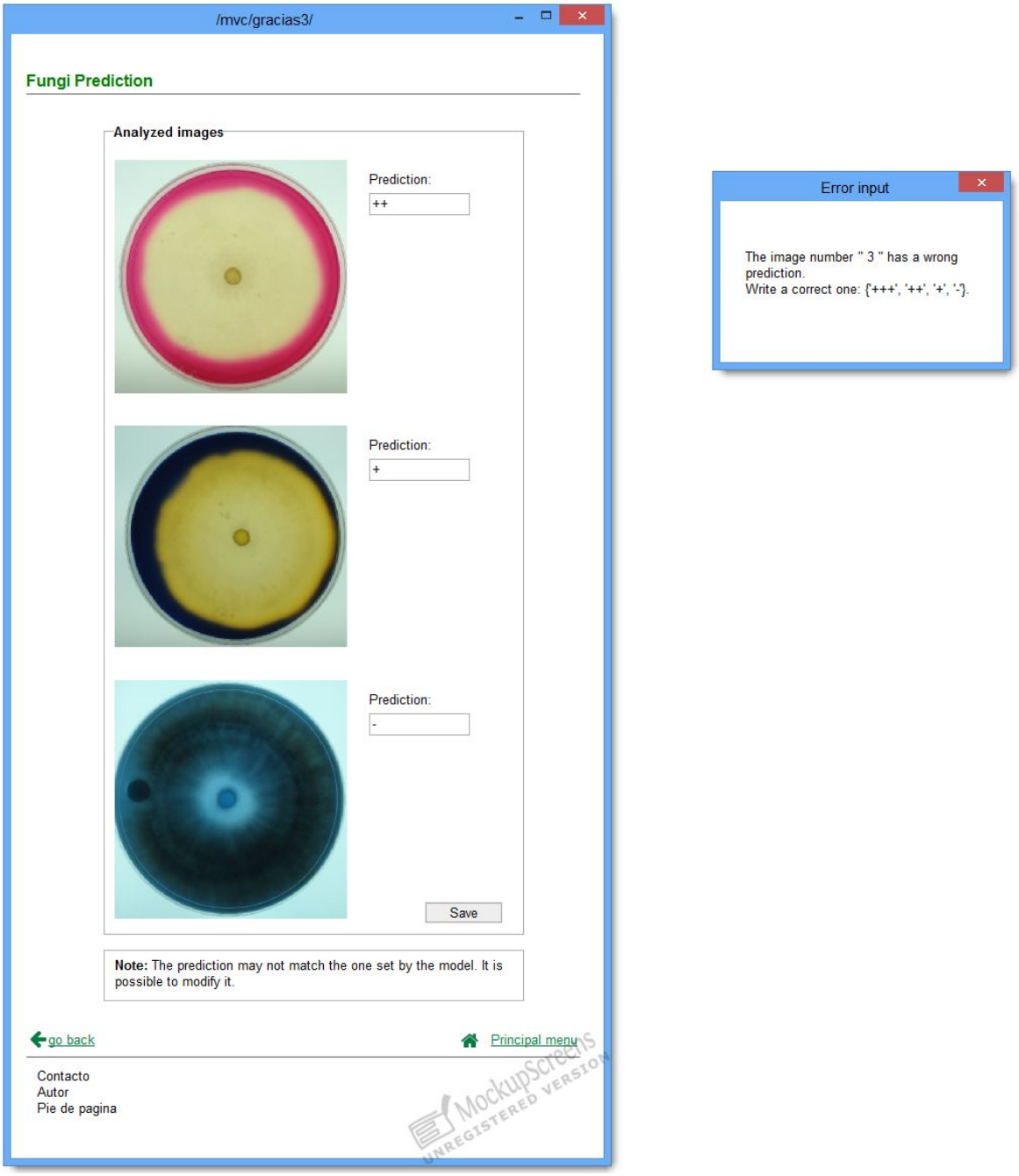


Figura 25: Prototipo de la vista predictiva tras el análisis de archivo zip de imágenes.

Las vistas predictivas tienen enlaces tanto a sus respectivas páginas anteriores, correspondientes a los formularios de subida de imágenes, como al menú principal. Además disponen del botón "*Save*" que permite almacenar los datos obtenidos en la base de datos.

4.2.5. Cambios en el modelo

Tras la primera semana del sprint se ha llevado a cabo la reunión con el cliente, en ella se ha decidido introducir cambios en la capa de persistencia. Los datos que se quieren almacenar, además de la URL de la imagen y la predicción, son los siguientes: el nombre de la muestra, el tinte utilizado y algunas anotaciones relevantes referentes a la predicción y estado del hongo. Los atributos nombre y tinte son requeridos, mientras que las notas pueden estar vacías. Es por ello que se ha tenido que volver a crear la tabla en la base de datos *fungi* con los nuevos campos especificados, y a su vez se ha tenido que modificar el modelo de nuestra aplicación,

creando una nueva clase donde se han añadido los nuevos atributos, todos ellos de tipo String. Se ha decidido crear una nueva clase llamada *ObjetoFungi* y no modificar la que teníamos, *ImagenyPrediccion,* ya que esta última sirve para mandar los datos de la predicción a la vista en la parte del controlador correspondiente al tercer método. Los cambios producidos en el código de la capa de persistencia han sido mínimos. Se puede ver la tabla *mvc_ObjetoFungi* en el *Anexo II.*

Tras la aparición de estos nuevos requisitos y datos a mostrar el alumno ha tenido que modificar los prototipos del diseño que se han presentado anteriormente (*ver apartado 4.2.4)*. Los nuevos cambios introducidos se han realizado principalmente en las vistas referentes a las predicciones de los distintos métodos, mientras que los prototipos de los formularios que enlazan las imágenes quedarían intactos. Para el nombre de la muestra, se ha decidido utilizar por defecto el nombre de la imagen sobre la que se realiza la predicción, este campo podrá ser modificado. Para el tinte habrá un input de tipo texto que en un principio aparecerá vacío, pero que posteriormente deberá ser rellenado. Tanto el campo nombre como el tinte son obligatorios y no deben estar vacíos. Sin embargo, para las anotaciones y observaciones debe aparecer un área de texto cuya cumplimentación no es obligatoria para el usuario.

En la *Figura 26* se muestra la vista predictiva, correspondiente al método que permite analizar una imagen. No se muestran los otros dos prototipos de los métodos restantes, ya que seguirán el mismo diseño.

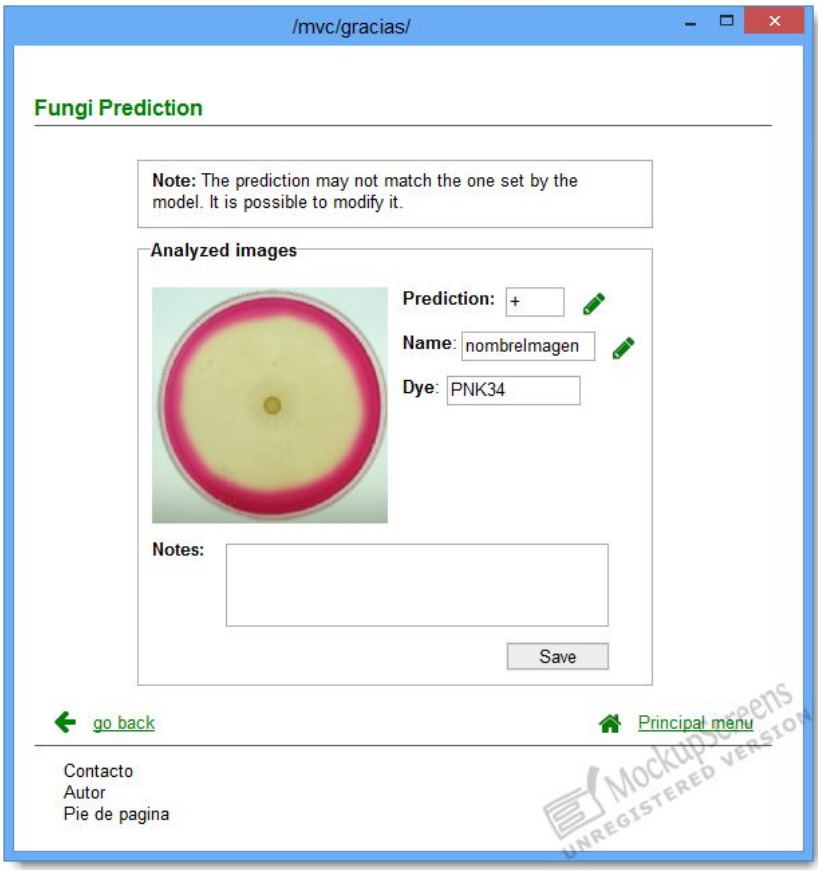


Figura 26: Nuevo prototipo de la vista predictiva tras el análisis de una imagen.

Hay que tener en cuenta que una gran parte del controlador ha tenido que ser modificada, ya que hay nuevos datos que hay que obtener de las peticiones y que se desean mostrar.

Una vez que el alumno ha realizado los cambios en el código del proyecto, se ha procedido a la evaluación de los prototipos por parte del cliente. Durante esta reunión, se han propuesto los siguientes cambios y mejoras:

-En el menú inicial se ha decidido enfatizar las tres opciones correspondientes a los tres métodos principales. Para ello, las cajas que enlazan dichos métodos deben de tener un tamaño más grande ocupando la mayor parte de la pantalla y estar centradas. Se trata de que los usuarios perciban tres alternativas posibles, y para ello deben existir tres objetos independientes, sin una jerarquía ni orden concretos, pero relacionados entre sí. Hay que eliminar la apariencia de unidad o bloque a la vez que parezcan opciones alternativas. Para ello se han tenido en cuenta los Principios de Agrupación de la Gestalt. En concreto, se han considerado el de semejanza, de cierre y de proximidad. Finalmente se ha decidido añadir un mensaje explicativo de bienvenida donde se presenten los métodos disponibles.

-Los enlaces de las vistas que redirigen tanto al menú inicial como a la página anterior, deben respetar todos el mismo formato y posición dentro de la página, buscando incrementar la consistencia. Además el nombre de los enlaces se debe modificar. Ahora el enlace de ir a la página anterior se llamará *"Previous page"* y el enlace de volver al menú principal, "*Back to menu".*

-Se propuso valorar la creación de un botón de ayuda en cada vista, como el que aparece en la *Figura 27*, de modo que al hacer clic se muestre en él la información relevante a las acciones que pueden realizarse en dicha vista, para evitar que la información resulte invasiva. Se ha decidido llevarlo a cabo, ya que se ha considerado que la información es de utilidad para usuarios novatos o aquellos que no usan frecuentemente la aplicación, además para los usuarios que la utilicen con frecuencia, dicha información puede resultar molesta. Para su realización se utilizará JavaScript.

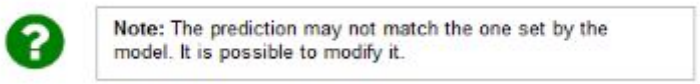


Figura 27: Prototipo del botón de ayuda.

-La predicción quedará estructurada a modo de tabla. En un lado aparecerá la imagen, mientras que en el otro, la información de la muestra analizada en el siguiente orden: predicción, nombre, tinte y notas.

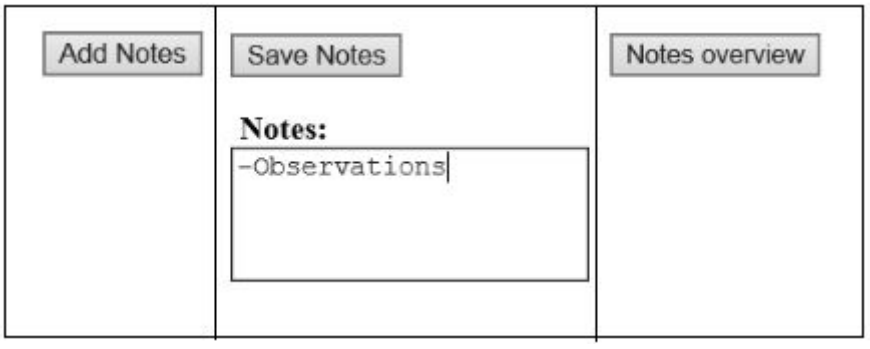


Figura 28: Prototipo de las notas. Es un botón que genera y esconde el área de texto donde escribir las observaciones.

-Para las notas se propuso la creación de un botón que permitiera la visualización dinámica de una caja de texto donde poder añadir las notas (*ver Figura 28)*. Se ha decidido llevar la idea a cabo, ya que según el cliente la funcionalidad para añadir observaciones no es muy utilizada entre los usuarios que usan la aplicación. Por tanto, la creación de este botón permitirá ahorrar el espacio que innecesariamente los *textarea* ocupaban. Para su realización se utilizará JavaScript.

-En la vista de la predicción correspondiente al método que permite analizar un conjunto de imágenes contenidas en un archivo zip, se ha decidido encuadrar cada imagen en un recuadro diferente, ya que son imágenes independientes unas de otras y el análisis se hace uno por uno, para, de este modo ayudar a que se perciban por separado.

-El botón de guardar los datos en la base de datos se situará fuera del *fieldset*, manteniendo la posición relativa en todas las vistas por igual. Y el nombre que aparece en el botón será *"Save data".*

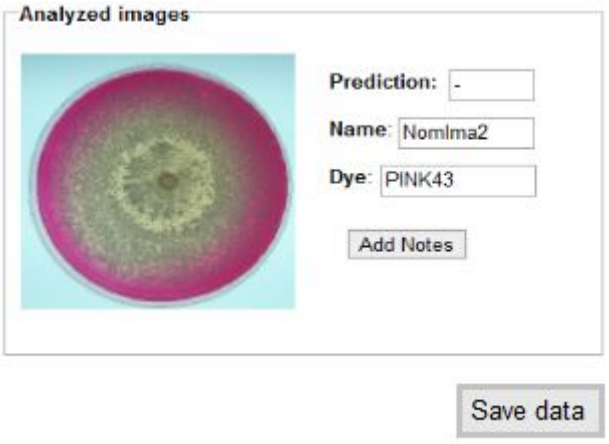


Figura 29: Prototipo botón "Save data".

Una vez realizado el análisis de los diseños, el alumno ha comenzado con la implementación de los mismos para finalizar el sprint. Éste realizará un diseño responsive de la aplicación web utilizando hojas de estilo CSS y HTML. Debido a que en este sprint no ha dado tiempo a implementar por completo las vistas, una parte de la implementación quedará pendiente para el siguiente sprint.

4.3. Sprint retrospective

Se han logrado realizar las siguientes tareas:

-Creación de la parte del controlador que permita realizar la predicción de las imágenes contenidas en un fichero zip.

-Construcción de la parte de la capa de persistencia que permita almacenar las predicciones de un archivo zip de imágenes.

-Diseño de las interfaces.

-Primera parte de la implementación del diseño de las vistas.

-Parseo del documento HTML obtenido como respuesta del modelo tras el análisis de un fichero zip.

Para su realización se han invertido un total de 72 horas.

Quedan pendientes para el siguiente sprint las siguientes tareas:

-Finalización de la parte del controlador que permita realizar la predicción de una imagen utilizando su imagen de control asociada.

-Finalización de la parte de la capa de persistencia correspondiente al método predictivo de una imagen y la correspondiente de control.

-Configuración de las url relativas de la aplicación. Esta última tarea la propuso el cliente al final del sprint con el fin de configurar adecuadamente la aplicación.

Capítulo 5: Sprint 4

5.1. Sprint planning

Tras haber realizado la reunión con el cliente y haber solventado las dudas acerca del segundo método que ofrece el modelo, se va a retomar la parte del trabajo que quedó pospuesta en el anterior sprint.

Las tareas previstas a lo largo del sprint son:

-Finalización de la parte del controlador que permita realizar la predicción de una imagen utilizando su imagen de control asociada.

-Finalización de la parte de la capa de persistencia correspondiente al método predictivo de una imagen y la correspondiente de control.

Además, posteriormente, se realizarán las siguientes tareas:

-Segunda parte de la implementación del diseño de las vistas.

-Configuración de las URLs relativas de la aplicación.

-Configuración del servidor de la universidad. Esta nueva tarea surge durante la reunión con el cliente con el fin de buscar un medio donde alojar el trabajo realizado por el alumno, y permitir que el cliente tenga acceso a la aplicación en todo momento.

5.2. Desarrollo

5.2.1. El controlador para el análisis de una imágen con su imagen de control

Vamos a retomar la parte del controlador correspondiente al método de analizar una imagen con su correspondiente imagen de control asociada, que habíamos aplazado al comienzo del sprint 3.

Tras la reunión con el cliente se han decidido que el alumno no modificará la parte del modelo correspondiente a las imágenes de control, además al no recibir las rutas tanto de la imagen de la muestra como la de control, en el JSON de la respuesta, se ha decidido mostrar dichas imágenes, en la vista de la predicción, desde la ruta correspondiente a nuestro proyecto. Esto es posible realizarlo, ya que las imágenes cuando son enviadas al modelo no se modifican, simplemente se devuelven tal y como se enviaron, por ello el cliente solo desea mostrarlas a los usuarios, es un trámite meramente informativo. Por tanto para poder configurar las rutas relativas de nuestra aplicación surge la siguiente tarea:

-Configuración de un directorio /media/ que permita albergar las imágenes sobre las que se realizará la predicción.

El directorio /media/ es el lugar de la aplicación donde se realiza la configuración de archivos estáticos, como son las imágenes y las hojas de estilo. Es decir, este directorio permitirá alojar

las fotos que se enviarán al modelo del cliente para obtener su predicción. Su configuración es muy rápida, debemos ir a la raíz de nuestro proyecto y crear una carpeta llamada "*media*", para ello en el fichero donde definimos las URLs, añadimos lo siguiente:

```
+static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Tras esto, vamos a modificar el fichero de configuración *settings.py* de nuestra aplicación, indicando dónde se guardará el directorio y la raíz, tal y como aparece a continuación:

```
MEDIA_URL = '/media/'
MEDIA_ROOT = 'media'
```

Una vez realizada la configuración, se debe invocar el *path* del directorio desde el template de la predicción y así poder mostrar las imágenes sobre las que se ha realizado el análisis. La manera de hacerlo utilizando Python embebido, es la siguiente:

```
<img src='{{ MEDIA_URL }}{{ imagenP }}' id="img" name="img" width="70%"/>
```

Tras realizar dichos pasos no se ha logrado que las imágenes se redireccionen bien y consecuentemente, no aparecen en la vista. El problema está en el archivo de configuración. Para solucionarlo, ha bastado con añadir dentro del apartado de templates, el comando correspondiente al proceso de contexto del directorio media.

5.2.2. La capa de persistencia para el análisis de una imagen con su correspondiente imagen de control

Para la finalización de la capa de persistencia se ha seguido la misma dinámica que en los sprints anteriores para los otros métodos del modelo. En primer lugar, se obtienen los parámetros de la petición con los que se realiza la creación de un objeto de tipo *ObjetoFungi,* establecido en nuestro modelo, y finalmente se llama al método *save()*.

Durante la implementación de esta parte surgieron dudas acerca de si era necesario almacenar información relevante de la imagen de control que acompaña a la imagen de muestra durante el análisis. Tras la reunión con el cliente se llegó a la conclusión de que no era necesario almacenar ninguna información sobre ella, ya que la característica principa de dicha imagen es el tipo de tinte, y como ambas imágenes deben tener el mismo tinte para poder realizar la predicción con este método, no hace falta guardarlo dos veces.

5.2.3. Implementación del diseño

Durante el final de la primera semana del sprint, el alumno ha continuado con la implementación de las vistas de la aplicación web siguiendo los prototipos evaluados y aceptados por el cliente, anteriormente presentados. Cabe señalar la importancia de JavaScript durante el diseño, ya que con él se ha conseguido añadir un botón para generar dinámicamente un bloque donde escribir notas si es necesario, también hemos definido la asistencia de ayuda y la gestión de cambio de los campos nombre y predicción.

Una vez finalizada la implementación de las vistas se procederá a una segunda evaluación por parte del cliente para ver si surgen cambios de última hora o si hay partes que aún necesitan mejorar.

5.2.4. Nueva funcionalidad de la aplicación

Tras haber finalizado, tanto la parte del controlador, como la parte de las vistas para los tres métodos que el modelo ofrece, y además, como el alumno dispone de tiempo para realizar mejoras en su aplicación web, el cliente ha decidido añadir un nuevo método con el que poder interactuar con su modelo, con el fin de aumentar la funcionalidad de la aplicación web.

El método consiste en analizar un fichero zip con imágenes de hongos dentro de sus respectivos discos de Petri, todas ellas con el mismo tinte en común. El zip puede contener imágenes del mismo tipo de hongo en distintas fases temporales y de crecimiento, o de hongos distintos. Es importante destacar que el archivo debe contener una imagen de control, es decir una imagen que sea solo tinte, y además el cliente ha especificado que su nombre debe ser "*control.jpg*". La predicción se realizará por parejas, es decir se realizará una petición al servicio web del cliente para cada imagen de muestra del zip acompañada con la de control, es como una iteración del segundo método.

Los objetivos de este nuevo método son, por un lado, aumentar la fiabilidad de las predicciones, ya que los resultados obtenidos tras hacer el análisis de una imagen con su correspondiente imagen de control son más fiables. Y por otro lado, la realización de predicciones en bloques, lo que ahorrará grandes cantidades de tiempo e interacciones entre el usuario y el modelo.

Tras la especificación del método por parte del cliente, surge la siguiente tarea a realizar:

-Diseñar e implementar un nuevo método que permita analizar un zip de imágenes a partir de una imagen de control. Esta tarea se va a dividir en la siguientes subtareas:

- Creación de la parte del controlador correspondiente al nuevo método.
- Creación de la parte de la capa de la persistencia correspondiente al nuevo método.
- Diseño de las nuevas vistas con *MockupScreen*.
- Creación e implementación de las vistas correspondientes al nuevo método.
- Modificación de la vista principal *Menú.html* para enlazar el nuevo método.
- Pruebas de test.

Nota: La estructura inicial de las vistas es exactamente igual que las correspondientes al método para analizar un fichero zip, desarrollado en el sprint 3. Hay una primera vista con un formulario que enlaza un zip y una segunda con un formulario que contiene la lista de predicciones de las imágenes con el mismo tinte y su información relevante, que se desea o no almacenar en la base de datos.

5.2.5. El controlador para el nuevo método de análisis

Para el desarrollo del controlador se ha seguido la misma dinámica que en los métodos anteriores, para ello se han implementado dos métodos: el primer método se encarga de obtener el fichero zip de imágenes cargado en el formulario de la primera vista y enviado mediante el método POST y con *enctype*, *multipart/form-data*. En segundo lugar se descomprime el zip en un nuevo directorio con el nombre del zip dentro de la carpeta /media/ del proyecto, seguido se emparejan las imágenes con la correspondiente de control y se envían las peticiones a la URL del modelo: '**http://193.146.250.57:8000/fungi_classification/predictFungiClassUsingOverfeatWithControl/**'.

Finalmente se obtienen las distintas respuestas elaboradas por el modelo en formato JSON, y se guardarán en la sesión los datos que queremos enviar a la vista correspondiente a la predicción. El segundo método es básicamente pasar el control a la vista de la predicción, en el cual hemos definido los atributos guardados en la sesión como variables para poder así mostrarlos en el template mediante Python embebido. En este método, también se desarrollará la parte correspondiente a la capa de persistencia.

5.2.6. La capa de persistencia para el nuevo método de análisis

El desarrollo de la capa de persistencia es exactamente igual que en el método para obtener la predicción de un fichero zip de imágenes (*ver apartado 4.2.3)*.

Durante la interacción con la capa de persistencia el alumno ha detectado un error general a la hora de guardar los datos, referente al almacenamiento de nulos en la columna descripción de nuestra tabla, ya que al crearla la hemos definido como NOT NULL. Esto se debe a que el bloque de texto para las observaciones, se crea de manera dinámica cuando se pulsa el botón *"Add Notes*", ya que recordemos que es un atributo opcional para la predicción. Por tanto, si no se hace clic en el botón de añadir notas, el área de texto no se genera, por lo que al enviar los parámetros del formulario para ser almacenados en la base de datos, el campo correspondiente al atributo descripción es nulo, y entonces aparece la excepción.

Para solucionarlo, el alumno ha decidido no generar el área de texto dinámicamente cuando se pulsa el botón, sino que ha añadido el atributo *hidden* sobre el contenedor del *textarea*, así el parámetro de la petición correspondiente a las notas no será nulo. Mediante Javascript ha sido posible modificar el valor de este atributo, lo que ha permitido mostrar y esconder el texto de las notas guardándose en todo momento. Esta funcionalidad favorece la organización y estructura de las vistas predictivas, y con su implementación concluimos con uno de los requisitos que el cliente propuso durante la evaluación de los prototipos de las interfaces.

Nota: Tras finalizar el controlador se ha continuado con el manejo de excepciones para los cuatro métodos implementados en la aplicación. Para ello se ha decidido crear una página de error a la cual se redirigirá, con su correspondiente mensaje de error, en caso de que ocurra una excepción o se viole alguna restricción.

5.2.7. Diseño de las interfaces

Para el nuevo método propuesto, el alumno ha realizado el correspondiente prototipo usando *MockupScreen*. La vista encargada de enlazar el zip con la imagen de control para ser enviada al modelo, es exactamente igual que para el tercer método dado por el modelo (*ver Figura 22)*.

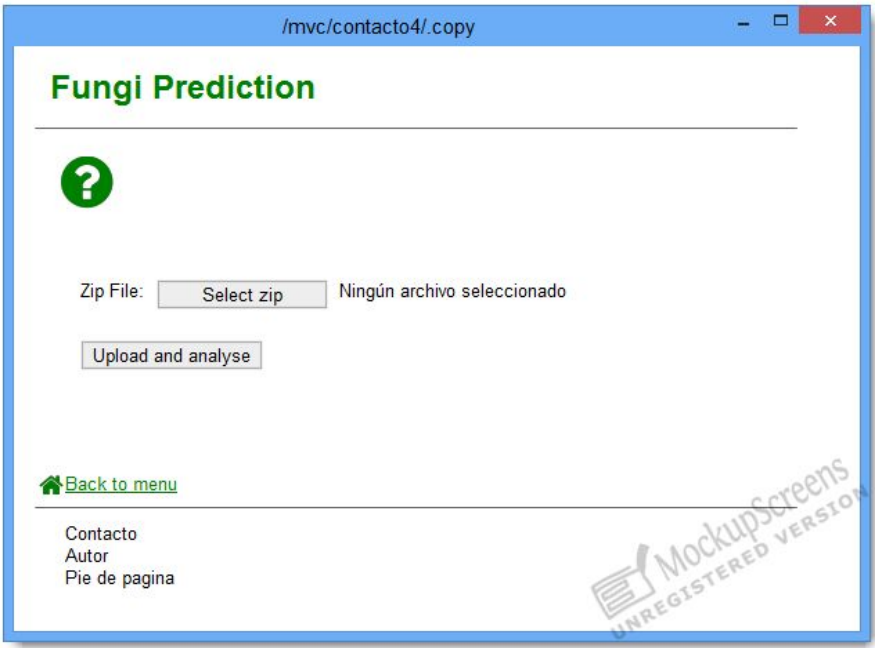


Figura 30: Prototipo del formulario para la vista correspondiente al análisis de un archivo zip de imágenes con una de control.

Para el diseño de la predicción se ha pensado que aparezca la imagen de control al principio con el tinte utilizado, ya que es común a todas las imágenes. A su vez, a la hora de insertar el nombre del tinte, se ha pensado en introducirlo una única vez y que mediante JavaScript se autocomplete en los demás campos.

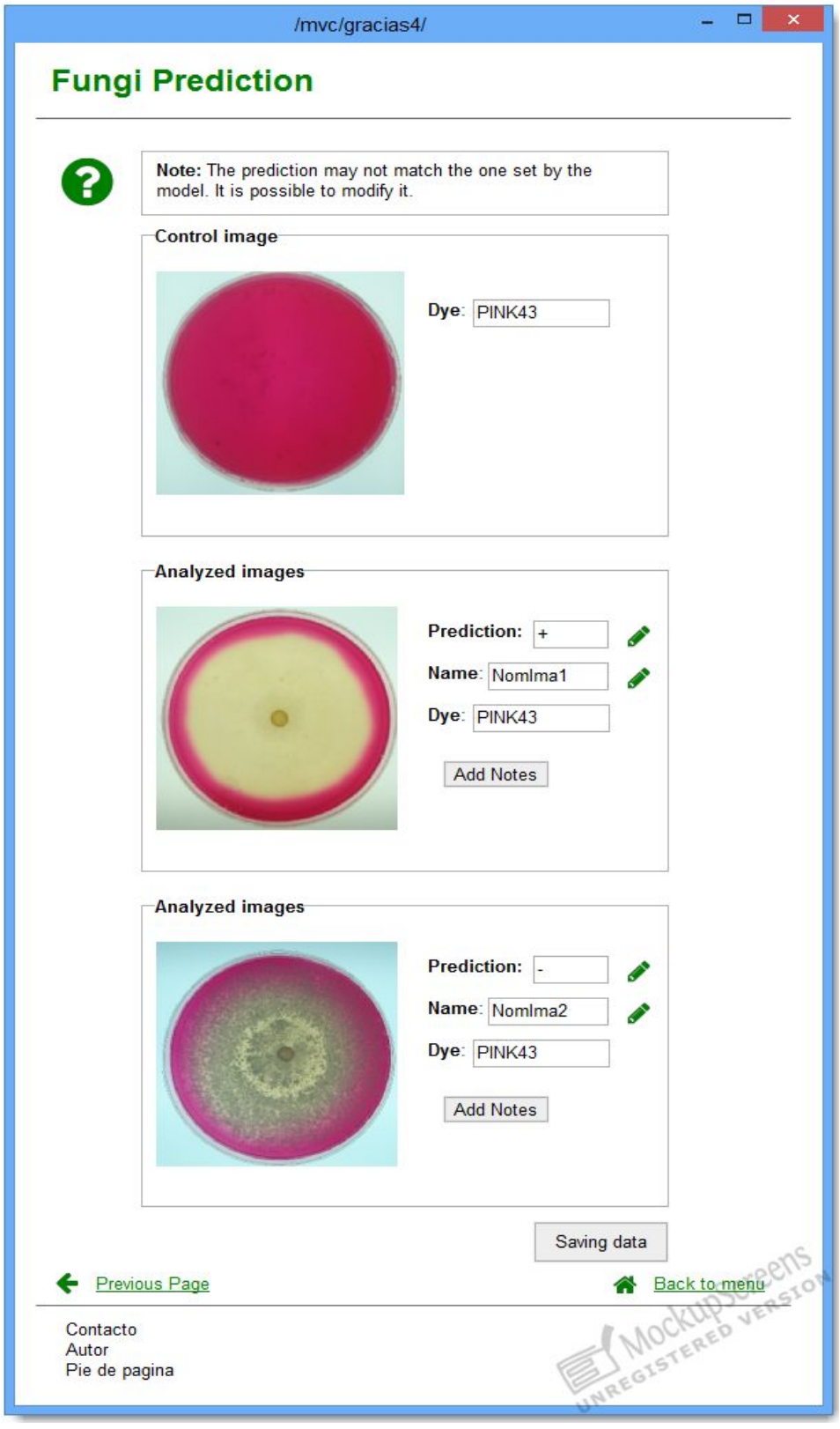


Figura 31: Prototipo de la vista predictiva tras el análisis de archivo zip de imágenes con una de control.

El diseño pensado para la página de error es el siguiente:

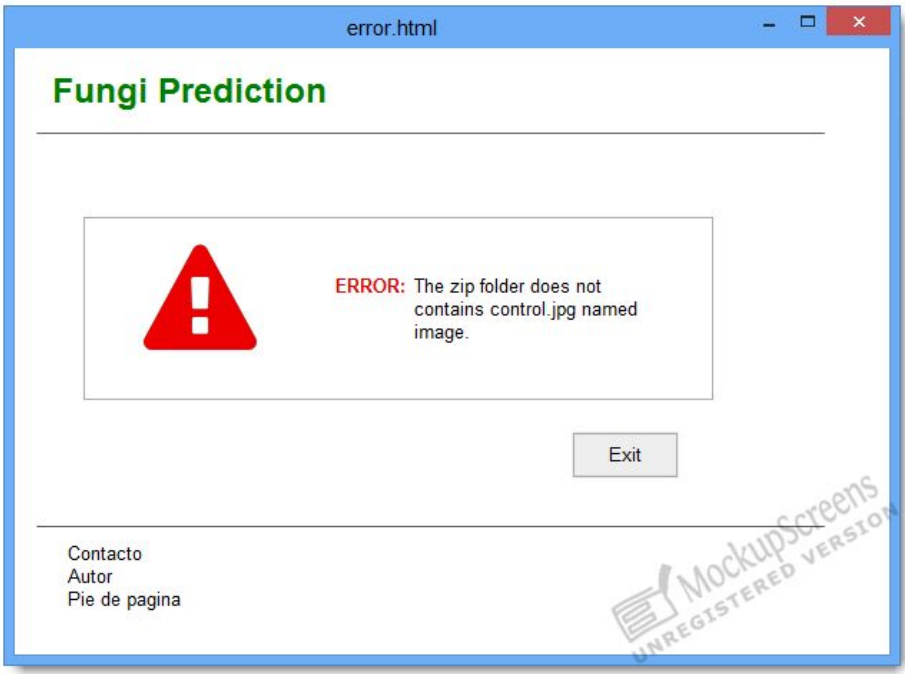


Figura 32: Prototipo vista de la página de error.

Una vez en la página de error, solo permitirá dirigirse al menú principal mediante el botón "*Exit*".

Finalmente se ha implementado el diseño siguiendo los prototipos anteriores, de manera que se ha podido reutilizar tanto el JavaScript como las hojas de estilo usadas anteriormente para los anteriores métodos.

Nota: Este último método añadido ha generado dudas en cuanto a la eficiencia del mismo, ya que para cada análisis se produce una llamada al modelo. Por tanto queda pendiente consultar con el cliente la posibilidad de añadir un servicio web, en el modelo dado, que implemente esta funcionalidad.

5.3. Sprint retrospective

Se han logrado realizar las siguientes tareas:

-Finalización de la parte del controlador que permita realizar la predicción de una imagen utilizando su imagen de control asociada.

-Configuración de las URLs relativas de la aplicación.

-Configuración de un directorio /media/ que permita albergar las imágenes sobre las que se realizará la predicción.

-Finalización de la parte de la capa de persistencia correspondiente al método predictivo de una imagen y su correspondiente imagen de control.

-Segunda parte de la Implementación del diseño de las vistas.

-Hacer un nuevo método zip que permita la predicción de sus imágenes utilizando su correspondiente imagen de control.

-Creación de la parte del controlador correspondiente al nuevo método.

-Diseño de las nuevas vistas con *MockupScreen*.

-Creación e implementación de las vistas correspondientes al nuevo método.

-Creación de la parte de la capa de la persistencia correspondiente al nuevo método.

-Modificación de la vista principal *Menú.html* para enlazar el nuevo método.

-Creación e implementación de una página de error.

Para su realización se han invertido un total de 70 horas.

Quedan pendientes las siguientes tareas:

-Configuración del servidor de la universidad.

-Pruebas de test.

Se ha decidido posponer estas últimas tareas, ya que su realización está ligada a la última parte del ciclo de vida del proyecto, correspondiente a la validación y el mantenimiento.

Capítulo 6: Sprint 5

6.1. Sprint planning

Se ha decidido realizar un sprint más debido a que el tiempo real, tanto para las tareas de creación de la parte del controlador, como para la capa de persistencia, ha sido menor que el planificado. Es por ello que el número de horas restante hasta alcanzar las 300, se dedicará a la mejora del diseño de las vistas, ya que conforma uno de los aspectos clave de este trabajo.

Por tanto la nueva y principal tarea a realizar durante el último sprint será:

-Mejoras en el diseño de las vistas.

Además las tareas que se realizarán para finalizar el sprint y consecuentemente el trabajo, serán:

-Configuración del servidor de la universidad.

-Pruebas de test.

6.2. Desarrollo

6.2.1. Mejoras propuestas por el cliente

Durante la reunión semanal, el cliente acompañado de un experto en usabilidad, evaluaron aspectos importantes del diseño presentado por el alumno que éste tiene que cambiar durante el desarrollo del nuevo sprint. A continuación, se explicarán las mejoras solicitadas por ambos.

En los formularios donde se cargan las imágenes y los ficheros zip, debe aparecer un correspondiente mensaje explicativo del método que se va a utilizar.

Actualmente la ayuda está implementada de manera que al pulsar el icono en forma de interrogación (*ver apartado 4.2.5)*, situado en la parte superior izquierda de las vistas, aparece un *textbox* con un mensaje específico de ayuda. Según el cliente, es una implementación que puede resultar intrusiva para el usuario. Se han propuesto varias ideas que el alumno debe valorar, una de ellas es mantener el icono anterior en otra posición distinta.

En las vistas predictivas, hay campos del formulario que son requeridos y otros que no lo son, por ello el alumno debe buscar la forma de distinguir unos de otros, ya que tal y como aparecen (*ver Figura 33),* el usuario no es capaz de diferenciarlos. Para el campo predicción se ha propuesto cambiar el *textbox* por un *combobox* no modificable con los valores posibles de la predicción (*ver Tabla 1*). Para el campo tinte se ha propuesto cambiar el *textbox* por un *combobox* en el que aparezcan los tintes más utilizados, pero esta vez que sea modificable, y así poder introducir el valor que el usuario desee. Con este diseño, se pretende minimizar los errores que pueda cometer el usuario al interactuar con la aplicación, incrementando así su usabilidad. Tras evaluar el prototipo del botón de notas que generaba el área de texto y la escondía (*ver apartado 4.2.5)*, el cliente no ha quedado muy satisfecho. Ha decidido volver al

diseño original, un área de texto pero con unas dimensiones menores. Por último, se ha propuesto buscar un sentido adecuado y un uso correcto para los enlaces en forma de lapicero, que permiten modificar el valor de la propiedad de solo lectura de los campos nombre y predicción, ya que tal y como está pensada esta funcionalidad, puede resultar confusa para el usuario final.

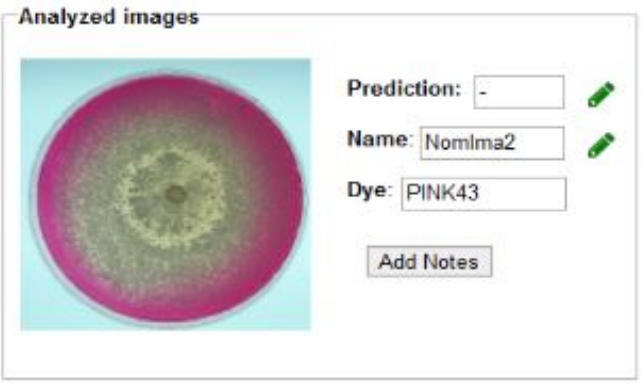


Figura 33: No hay diferencia entre los campos requeridos y no requeridos.

Por otro lado, el cliente ha propuesto cambiar la estructura de cómo se muestran las predicciones obtenidas, de manera que se pueda aprovechar el tamaño de la pantalla. El fin de este cambio es permitir que los usuarios puedan ver el mayor número de predicciones en una pantalla, para así poder realizar comparaciones a la hora de decidir si cambiar o no la predicción asignada por el modelo.

Para los métodos que analizan archivos zip de imágenes, el cliente ha añadido un nuevo requisito. Éste quiere que de todas las predicciones obtenidas tras el análisis de las imágenes contenidas en el zip, el usuario pueda elegir cuáles guardar en la base de datos y cuáles no.

El cliente ha considerado que cuando se obtiene la vista predictiva para el método que analiza un fichero zip de imágenes con una imagen de control, tanto la imagen de control como el tinte deben permanecer siempre visibles para el usuario.

Finalmente, como resultado de la evaluación de la interfaz realizada por el experto en usabilidad, el alumno debe mejorar el diseño de la aplicación web. Una de las sugerencias planteadas por el experto es que el alumno revise el diseño de las vistas para intentar extraer un esquema de diseño al que se adapten los cuatro métodos de análisis. También sugirió al alumno revisar distintas aplicaciones sobre temas de biología para que se haga una idea del tipo de diseños que suelen utilizarse en este ámbito (diseño relacionado con la experiencia de los usuarios finales).

6.2.2. Implementación del diseño

En primer lugar se ha decidido modificar el aspecto general de todas las vistas, para ello se ha eliminando el pie de página, ya que daba muchos problemas a la hora de ajustar el contenido de la pantalla. En su lugar se ha creado un menú en forma de barra horizontal fija como el que aparece en la *Figura 34*, situado en la parte superior de todas las vistas, justo debajo del nombre de la aplicación.

Figura 34: Menú horizontal común a todas las vistas.

Las opciones que el menú dispone son las siguientes:

-*Home*: permite al usuario acceder al inicio desde cualquier punto de la aplicación cuando le interese.

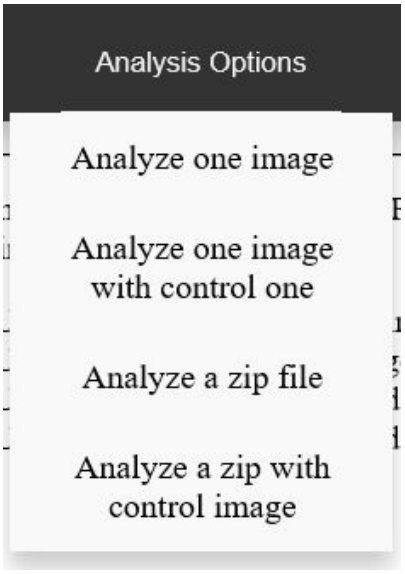


Figura 35: Menú desplegable con las opciones de análisis.

-*Analysis Options:* este apartado genera un menú desplegable (*ver Figura 35)* formado por enlaces a las cuatro opciones predictivas que ofrece la aplicación: analizar una única imagen, analizar una imagen con una de control, analizar un zip o analizar un zip con una imagen de control. Haciendo clic en cualquiera de ellas nos permite acceder al formulario de carga de archivos correspondiente. Se barajó la posibilidad de colocar un menú exclusivo para esta funcionalidad en la parte izquierda de cada vista, pero en el caso de analizar varias imágenes restaría espacio para mostrar las predicciones, por lo que se decidió solo incluirlo en la barra menú. Además, en este apartado, aparece coloreado el recuadro correspondiente al método que se esté utilizando en dicho momento, esta funcionalidad permite al usuario ubicarse dentro de la aplicación (*ver Figura 36),* favoreciendo la navegabilidad de la misma.

-*Contact*: permite al usuario ponerse en contacto con el gestor del sitio web, a través de un correo electrónico. Faltará introducir el correo del cliente en la implementación del enlace.

-*About*: permite acceder a la información relevante de la aplicación.

-*Tutorial*: este apartado contiene información de ayuda útil para aquellos usuarios novatos o con muy poca práctica en el uso de esta aplicación. Sería deseable que esta página, a parte de albergar tutoriales con imágenes y extensas explicaciones, también

contuviera algunos vídeos sobre el funcionamiento de la aplicación.

Figura 36: Menú desplegable que indica al usuario que se encuentra en el método que permite analizar un fichero zip.

La barra menú es uno de los aspectos que facilita la navegabilidad de nuestra aplicación. Otro aspecto influyente, ha sido el uso de *breadcrumbs* o migas de pan en cada una de las vistas (*ver Figura 37)*. Esto permite al usuario conocer, en todo momento, la ruta que ha seguido desde el menú inicial hasta la pantalla actual en la que se encuentra.

La página actual en la que se sitúe el usuario, aparecerá en color verde intenso, mientras que las páginas precedentes tendrán un color apagado y además servirán como enlaces para permitir al usuario volver sobre sus pasos.

Figura 37: Migas de pan o breadcrumbs.

También se pensó en introducir un mapa del sitio, pero esta idea fue descartada debido a la reducida extensión de la aplicación web.

Hasta ahora el alumno ha implementado dos vistas para cada método del modelo y una para el menú principal, haciendo un total de 9 vistas. Tras analizar con el cliente las vistas, se ha percibido que existe un esquema de diseño común en muchas de ellas. Se han agrupado las vistas implementadas en tres grupos: en primer lugar tenemos la vista correspondiente al menú principal, seguido tenemos las vistas de los formularios que permiten subir archivos al modelo para ser analizados y finalmente las vistas de los formularios con los resultados predictivos tras el análisis. Es por ello, que se ha propuesto al alumno implementar cada grupo de vistas comunes en una única vista, con el fin de aportar un diseño estructurado y común a toda la aplicación, buscando siempre la coherencia y la consistencia de la misma. Consecuentemente se reducirán el número de hojas de estilo unificándolas en un menor número de éstas, y además se deberá modificar la parte del controlador. Este nuevo giro en la implementación favorecerá el mantenimiento y futuras modificaciones que se realicen sobre las vistas.

A continuación se va a analizar cada una de las vistas, explicando sus partes y funcionalidades

más destacadas.

1. Menú principal

El menú principal está formado por dos partes, en primer lugar aparece un mensaje introductorio explicando los distintos métodos de análisis que el servicio web ofrece al usuario, y en segundo lugar, en el centro de la página, aparecen las cuatro opciones que enlazan a las respectivas vistas que permiten cargar los archivos que se deseen analizar.

Javascript nos ha permitido modificar los bloques como si fueran botones, por tanto si pulsamos en cualquier parte del recuadro inmediatamente se realiza la redirección a la respectiva vista siguiente.

En el *Anexo II* aparecen las capturas finales de la aplicación web.

2. Estructura de los distintos formularios

El primer grupo de vistas, es aquel que contiene un formulario que permite subir ficheros, tanto imágenes como archivos zip, y enviarlos al modelo para su correspondiente análisis. El alumno ha procedido a la unificación de estas vistas en una sola siguiendo la estructura de diseño que aparece en la *Figura 38.*

Figura 38: Estructura común a los formularios de carga de imágenes.

El formulario de carga se creará dinámicamente dependiendo del método elegido. Se pueden diferenciar tres tipos de formulario para cargar imágenes: con un único enlace para subir una imagen, con dos enlaces para subir una imagen y su correspondiente imagen de control y con un enlace para subir un archivo zip, que contenga o no imagen de control.

Siguiendo las especificaciones del cliente se ha introducido, en la parte superior de cada formulario, un texto introductorio explicando el método a utilizar.

El segundo grupo de vistas, es aquel que muestra el formulario con los datos predictivos

obtenidos a partir del análisis del modelo. Es importante destacar que el diseño de tres de dichas vistas, correspondientes a los casos de una única imagen, dos imágenes una de ellas de control y un archivo zip, pueden verse como un caso particular del diseño de la vista más compleja, aquella correspondiente a la predicción de un archivo zip con una imagen de control.

Por tanto, el alumno ha procedido a la unificación de estas vistas en una sola, partiendo del diseño de la vista más compleja, como bien hemos dicho, aquella correspondiente al método zip con una imagen de control. Ésta sigue la estructura de diseño que aparece en la *Figura 39*.

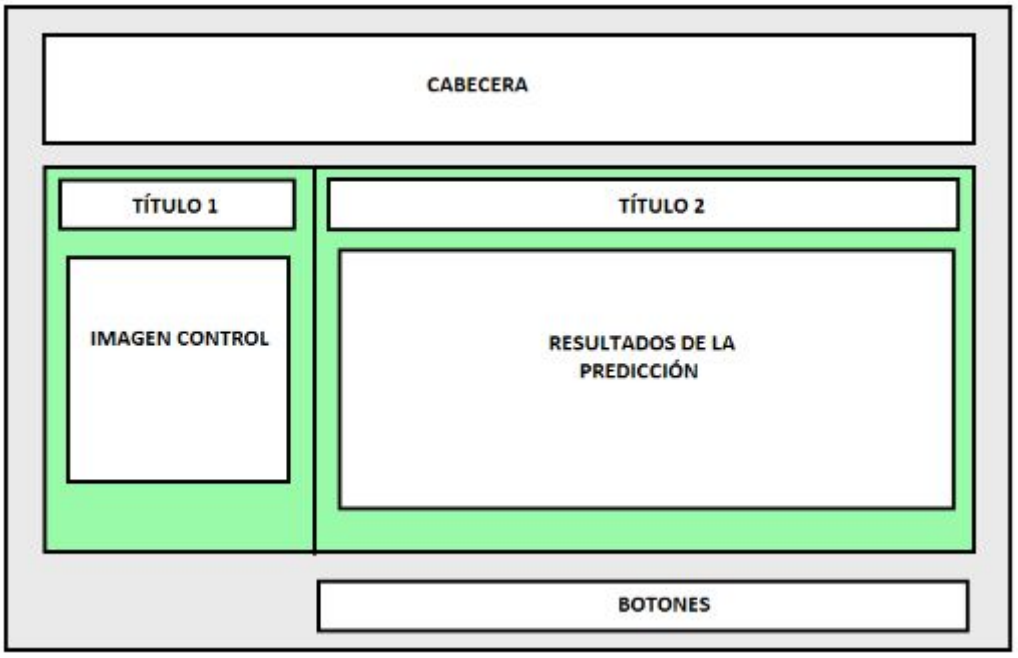


Figura 39: Estructura común a las vistas predictivas.

Los detalles de las vistas que diferencian cada método del modelo, se generarán de manera dinámica con la utilización de Python embebido, lo que permitirá estructurar una única vista de cuatro maneras diferentes.

La parte de la cabecera es común a este grupo de vistas, lo que variará, según el método a utilizar, será la manera de mostrar el contenido y los botones a utilizar para gestionar dicho contenido.

La parte izquierda solo aparecerá en aquellos métodos que utilizan imagen de control, ya que esta zona está reservada exclusivamente para mostrar dicha imagen (*ver Figura 40)*. La imagen de control, en el caso de que haya, siempre debe aparecer visible, por tanto hemos decidido colocarla a la misma altura que la imagen o imágenes analizadas, con el fin de que el usuario pueda realizar una comparación constante en todo momento.

Es importante destacar que para el método que permite analizar un fichero zip con una imagen de control, también se mostrará en todo momento el tinte, ya que va a ser un campo común a todas las imágenes analizadas contenidas en el zip.

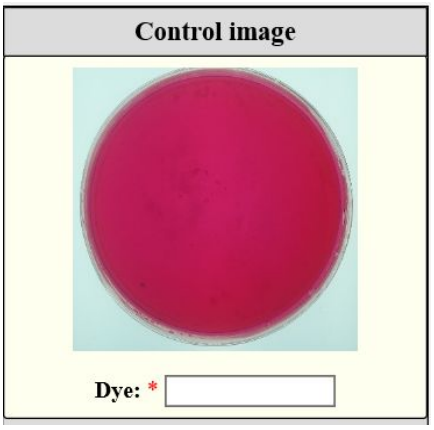


Figura 40: Imagen de control y tinte siempre visibles en el método zip con una imagen de control.

En la parte derecha aparecen organizadas en una tabla, las imágenes analizadas con sus respectivos campos de datos asociados. En el caso de haber elegido un método que no use imagen de control, la amplitud de este campo se expande hasta adaptarse al ancho de la pantalla. El fin de este diseño es permitir al usuario ver el mayor número de predicciones a la vez.

3. Sección de Botones

Se han agrupado los botones asociados al formulario en la parte inferior a la zona donde se sitúa el resultado del análisis predictivo. Es un área siempre visible para el usuario y cercana al último campo del formulario, bajo la parte derecha donde aparecen las predicciones.

Hay tres botones, alguno de ellos, dependiendo del método que se utilice, tendrá un nombre u otro (*ver Figura 41)*. Se ha intentado buscar un nombre adecuado para la acción que realiza, intentando no asignar nombres generales y buscando el reconocimiento por parte del usuario más que el recuerdo.

-Botón "*Restore"*: permite restaurar los valores por defecto que los campos predictivos tenían tras el análisis del modelo. Esto posibilita al usuario volver a un estado inicial donde poder rellenar los campos de nuevo a partir del análisis de manera rápida y eficaz.

-Botón *"Back to Analyse"*: permite al usuario volver a la página anterior correspondiente con la vista que contiene el formulario de carga. Este botón influye en el aspecto de navegabilidad de la aplicación.

-Botón "*Save Data"*: este permite al usuario almacenar los datos predictivos en la base de datos. En el caso de los métodos que analizan imágenes contenidas en ficheros zip, el nombre que hemos decidido poner al botón es "*Save Selection"*, en relación a la funcionalidad de elegir los datos a almacenar.

Figura 41: Sección de botones dependiendo de si el método utiliza un fichero zip o no.

Es importante el orden de los botones, por ello se ha decidido situar a la derecha el botón de mayor importancia y cuya acción asociada prima frente a las demás, es el caso del botón "*Save Data*" *("Save Selection"* en el caso de métodos que analizan ficheros zip de imágenes*)*. Mientras que a la izquierda de éste aparecen los otros dos botones cuyas acciones son secundarias.

4. Diseño de las predicciones

Cada predicción realizada por el usuario formará parte de la celda de una tabla. En cada celda se mostrará en el lado izquierdo la imagen del hongo analizada, y en lado derecho, a la misma altura, los datos obtenidos tras la predicción y los campos que el usuario debe rellenar previamente a su almacenamiento en la base de datos. En la *Figura 42*aparece la estructura del diseño de la predicción asociada a una imagen.

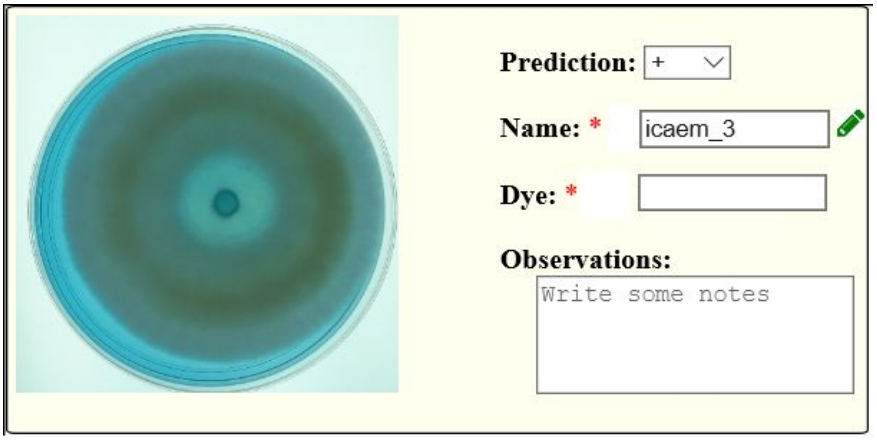


Figura 42: Diseño de la predicción asociada a una imagen.

Se observa que tanto los campos de texto como las etiquetas están alineadas y agrupadas de manera vertical y horizontal. Con el fin de que el usuario revise y añada la información en un orden determinado, de arriba abajo, y además con el fin de jerarquizar la importancia de los datos. Es por ello, que en la parte superior se encuentra el campo predicción, que es el dato más importante de cada análisis y sobre el que está basada la aplicación web.

Para destacar los campos obligatorios que el usuario debe rellenar antes de guardar los datos, se ha utilizado un asterisco de color rojo siguiendo reglas de diseño habituales y entendidas por todos los usuarios.

Cabe recordar que tras el análisis, los campos predicción y nombre siempre tienen un valor por defecto, que a decisión del usuario, pueden ser cambiados. Los demás campos aparecerán vacíos tras la predicción. Por ello se ha decidido implementar su diseño de la siguiente manera:

-Al campo predicción se ha decidido asociar un *combobox (ver Figura 43*) con los posibles valores predictivos que el modelo asocia a una imagen, estos aparecen en la *Tabla 1*. Es un campo que nunca va a estar vacío.

Figura 43: Combobox con las predicciones posibles.

El color rojo indica que puede modificarse, pero una vez que el usuario haga clic en el *combobox* con intención de modificar, éste se volverá blanco hasta el final de la sesión.

-Tras el análisis, el campo nombre tiene por defecto el nombre de la imagen y es de solo lectura, es decir, no modificable. Se ha decidido añadir dicha propiedad en el caso de que el usuario no quiera modificar el nombre que la aplicación pone por defecto a la imagen.

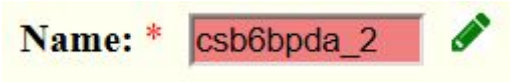


Figura 44: Campo nombre tras el análisis, solo lectura.

En el caso de que el usuario quiera modificar el valor, hay disponible un icono con la forma de un lapicero (símbolo habitual de edición), de modo que al hacer clic sobre él, la propiedad solo lectura del cuadro de texto se deshabilita, pudiendo así el usuario modificar el valor (*ver Figura 45)*.

Figura 45: Campo nombre en estado de edición.

La propiedad de solo lectura va asociada al color rojo del cuadro de texto y el verde del icono del lapicero (*ver Figura 44)*, por tanto al deshabilitar dicha propiedad para realizar la modificación, el fondo del *textbox* se volverá blanco y el color del lapicero más tenue. Es importante destacar que una vez que se han pulsado el icono, los colores asociados a la modificación se mantendrán con el fin de indicar al usuario que el campo asociado a dicha predicción, ha sido modificado.

De este modo, manteniendo la ausencia de color hasta el final de los campos modificados, el usuario está informado del estado de la aplicación.

-Tanto el campo nombre como el campo tinte son cuadros de texto que tienen la propiedad de autocompletado. Esta implementación ha sido valorada positivamente por el cliente, es por ello, que éste se ha retractado en la idea de usar un *combobox* con los tintes más utilizados.

-Finalmente el campo observaciones se ha implementado como un área de texto en la que el usuario pueda escribir sus anotaciones sobre la imagen obtenida.

5. Funcionalidad para los métodos que analizan un fichero zip

Se ha implementado la funcionalidad que permite al usuario, tras el análisis de un archivo zip, realizar una selección de aquellas imágenes y sus respectivos datos que desee almacenar, cumpliendo así con el requisito propuesto por el cliente en la última reunión.

A cada imagen analizada se le ha asociado un *checkbox*; éste se sitúa en la parte superior derecha de cada celda sobre los datos del análisis de cada imagen, tal y como aparece en la *Figura 46*.

Se ha decidido que todos los *checkbox* aparezcan marcados tras el análisis. Se entiende que la opción más probable es aquella en la que el usuario que carga un zip de imágenes a analizar,

va a querer almacenar todas las predicciones obtenidas.

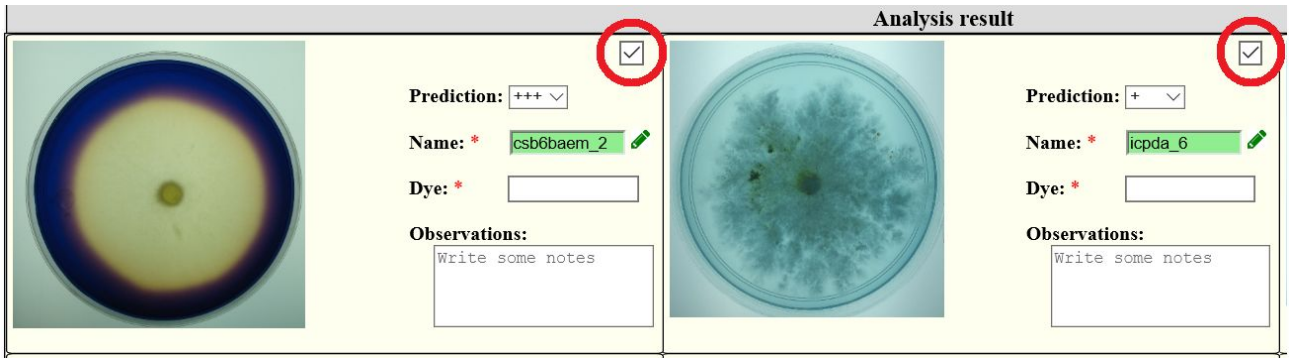


Figura 46: Checkbox en la parte superior de cada predicción.

En el momento en el que un usuario desmarca un *checkbox* de una predicción determinada, los datos y campos asociados de está quedarán inmediatamente deshabilitados *(ver Figura 47),* cambiando el color de estos e impidiendo, a su vez, que se modifiquen. Esta es la manera en la que la aplicación señala al usuario que una predicción no se va a almacenar una vez guardados los datos.

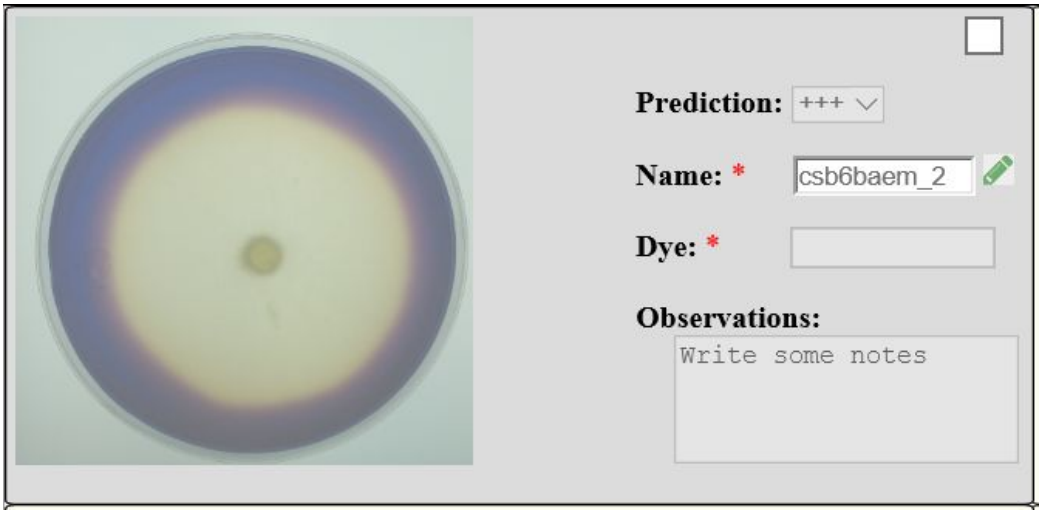


Figura 47: Predicción aleatoria de una imagen de un fichero zip deshabilitada,

A su vez se ha creado un *checkbox* general con la etiqueta "*Select all:*", que nos permite seleccionar y deseleccionar todas las imágenes a la vez (*ver Figura 48)*. Este input irá situado en la zona de botones a la izquierda de todos ellos.

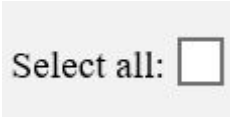


Figura 48: Checkbox general, permite seleccionar y deseleccionar todas las predicciones a la vez.

6. Implementación de la ayuda

Se ha intentado mantener el icono de ayuda que el alumno había implementado en un primer lugar, buscando una posición adecuada para éste. Finalmente, tras mostrar el resultado obtenido al cliente, no ha quedado satisfecho, por lo que el alumno ha tenido que buscar nuevas formas de implementar la ayuda.

Tras consultar en diferentes aplicaciones sobre temas de biología, se ha observado que la mayoría de ellas usan tooltips. Es por ello que vamos a utilizar estas herramientas en nuestra aplicación.

Los tooltips aparecerán al pasar el cursor sobre determinadas zonas de los formularios.

-En las vistas que contienen los formularios para enlazar los archivos a analizar de cada método, aparecerá un mensaje explicativo, indicando al usuario los pasos necesarios que debe seguir para obtener el análisis predictivo (*ver Figura 49)*. El mensaje aparece al posicionarnos en cualquier parte del formulario.

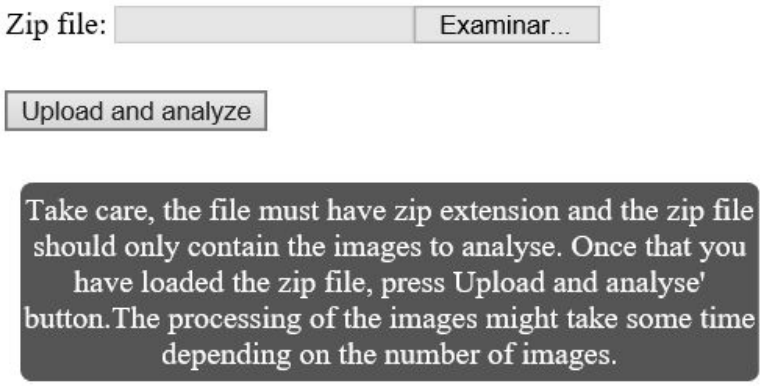


Figura 49: Diseño de los tooltips en las vistas de los formularios de carga de imágenes.

-En las vistas predictivas, cuando nos situamos sobre los campos: predicción, nombre y tinte, aparece un mensaje específico debajo de cada uno de ellos (*ver Figura 50)*.

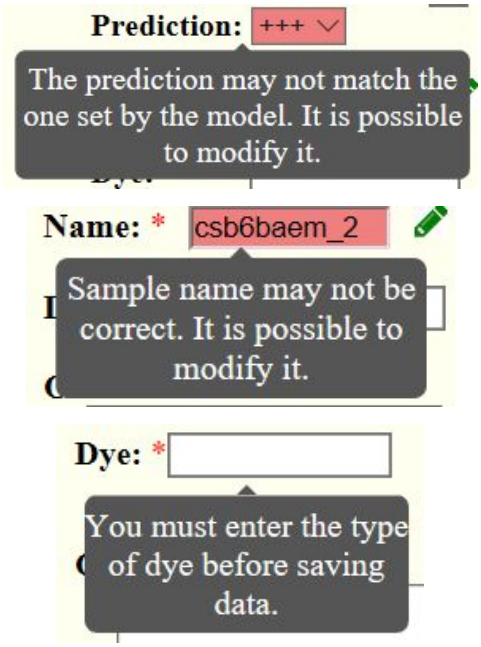


Figura 50: Diseño de los tooltips en las vistas predictivas, uno para cada campo.

7. Gestión de errores en los formularios

Se ha realizado anteriormente una página de error para el manejo de las excepciones. Sin embargo, el cliente hizo hincapié en que los errores de los formularios debían aparecer en

posiciones cercanas a los campos donde dicho error se producía, favoreciendo así la usabilidad de nuestra aplicación.

Las comprobaciones que se han realizado son las siguientes:

-Los archivos que el usuario desee analizar deben situarse en la carpeta /mvc/media/ de la aplicación, esto facilitará al usuario en el futuro, en caso de que quiera recuperar las imágenes analizadas.

-Los métodos correspondientes al análisis de imágenes comprobarán que los archivos cargados son imágenes y no otro tipo de ficheros. Las extensiones que el formulario de carga soporta son: jpg, jpeg, gif, png y bmp.

-En los métodos correspondientes al análisis de las imágenes contenidas en un archivo zip, se comprobará que los archivos cargados son ficheros con extensión zip. Además para el método correspondiente al análisis de un zip con una imagen de control, se ha comprobado que contenga una imagen llamada "*control.jpg*".

Los mensajes de error aparecerán dentro de un cuadro en rojo, como el que se muestra en la *Figura 51,* sobre el formulario de carga.

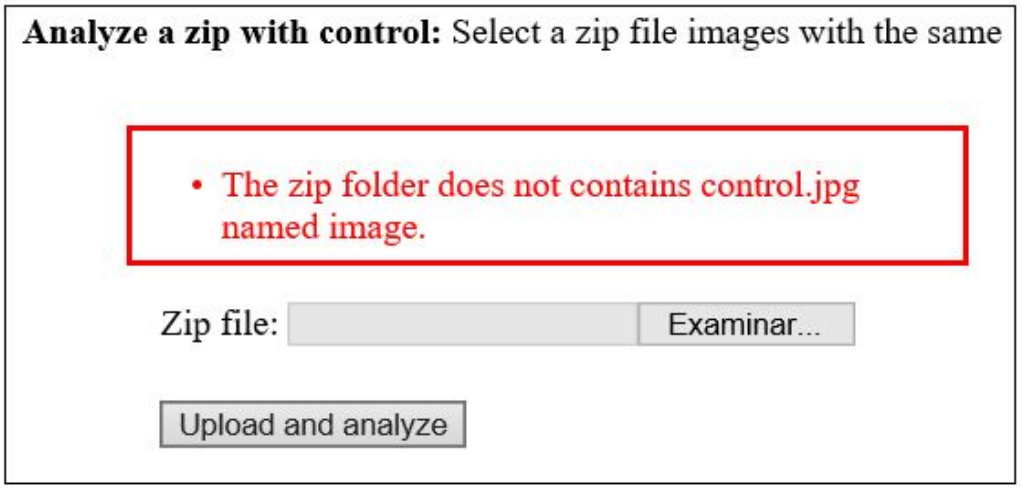


Figura 51: Ejemplo de un mensaje de error en el formulario de carga del método zip con una imagen de control.

-En las vistas predictivas, los campos nombre y tinte son obligatorios, por tanto si el usuario envía el formulario para guardar los datos en la base de datos antes de tiempo, aparecerá un mensaje de error en el propio campo avisando al usuario de que dichos campos no pueden estar vacíos. Además el cuadro de texto aparecerá remarcado en rojo indicando el error. El fin de esta implementación es facilitar al usuario la localización del error dentro del formulario.

6.2.3. Cierre del proyecto

Para finalizar el proyecto se han realizado las dos tareas que quedaban pendientes en la planificación del sprint.

En primer lugar se han realizado varias pruebas de test con usuarios reales, algunos de ellos biólogos. La calificación dada por estos ha sido satisfactoria.

Y en segundo lugar se ha configurado la aplicación en el servidor de la universidad siguiendo las instrucciones dadas por el cliente.

Finalmente se ha logrado que la aplicación funcione, y a su vez tenga un diseño responsive, para los navegadores: Microsoft Edge, Google Chrome y Firefox.

6.3. Sprint retrospective

Se ha logrado realizar las siguientes tareas:

-Mejoras en el diseño.

-Configuración del servidor de la universidad.

-Pruebas de test

6.4. Gestión del proyecto

A continuación se muestra la *Tabla 2* con la comparativa entre las horas planificadas y las horas reales para la realización cada sprint.

Sprint	TIempo planificado	Tiempo real	Desviación
Sprint 1	75 h.	60 h.	-20%
Sprint 2	75 h.	65 h.	-13.3%
Sprint 3	75 h.	72 h.	-4%
Sprint 4	75 h.	70 h.	-6.7%
Sprint 5	0 h.	23 h.	
Total	300 h.	300 h.	

Tabla 2: tiempo planificado contra tiempo estimado de cada sprint.

Conclusiones

El propósito de este trabajo ha sido realizar una aplicación web siguiendo el patrón de diseño MVC en Python.

En cuanto a los conocimientos adquiridos por el alumno, destacan el grado de aprendizaje obtenido del uso del lenguaje Python en el ámbito del desarrollo de aplicaciones web. A su vez, el alumno ha entendido la importancia de tener en cuenta los aspectos relacionados con la usabilidad a la hora de desarrollar una aplicación.

En cuanto al nivel de la planificación ágil, se podría decir que se han respetado las tareas propuestas para cada sprint, es importante destacar que los tiempos asignados para realizar cada tarea durante la planificación, han dispuesto de una gran holgura. Ésto ha permitido realizar las nuevas tareas que han ido apareciendo a lo largo del desarrollo del proyecto.

El resultado final ha sido satisfactorio, debido a que se ha desarrollado una aplicación de calidad, dando importancia, a su vez, a la calidad del diseño elaborado.

7.1. Posibles mejoras y extensiones

Han quedado pendientes varias mejoras en el diseño de la aplicación, que por motivos de tiempo no se han podido implementar, y que en fases futuras podrán desarrollarse.

Las mejoras concretas para poder terminar de refinar la interfaz son:

-Cambiar el color al tick ✔ de los *checkbox* a verde.

-Conseguir que la aplicación web se ejecute en más navegadores.

-Realizar un apartado de configuración que permita desactivar los tooltips de la aplicación.

Las extensiones que se han pensado son las siguientes:

-Realizar una correcta gestión de usuarios para que cada usuario pueda realizar sus operaciones sobre sus análisis.

-Se ha dejado preparada la base de datos para poder reentrenar al modelo. Es importante destacar la importancia de la gestión de usuarios que se debe implementar, ya que hay que tener cuidado con los usuarios que realizan las predicciones a partir de las cuales el modelo pueda aprender.

Bibliografía

[1] Python Software Foundation: https://docs.python.org/

[2] Django oficial page: https://www.djangoproject.com/

[3] Python requests: http://docs.python-requests.org/en/master/

[4] Django documentation: https://docs.djangoproject.com/es/

[5] MySQL in Django: http://blog.johnserrano.co/mysql-con-python-django/

[6] BeautifulSoup documentation: https://www.crummy.com/software/BeautifulSoup/

[7] MockupScreen oficial page: http://www.mockupscreens.com/

[8] Información de HTML y CSS: https://www.w3schools.com/html/html_css.asp

[9] G. Izquierdo, Francisco. *Programación de aplicaciones web*. Universidad de La Rioja. Curso 2016/2017.

A-Anexo I: Creación de un proyecto en Django

Se puede crear un proyecto de dos maneras distintas, una desde PyCharm: *File* >> *New Project* >> *Django*, y otra desde la *cmd* introduciendo el comando que aparece en la *Figura 52.*

```
C:\Users\ignac>django-admin.py startproject blog .
```

Figura 52: Creación de un proyecto Django a través de la cmd.

Es importante incluir el "." al final del comando para indicar al script que instale Django en el directorio actual. Cabe destacar que *django-admin.py* es el script encargado de generar el directorio de trabajo con los archivos por defecto que contiene un proyecto Django. Tras escribir el comando, el directorio creado tiene la estructura que aparece en la *Figura 53.*

```
directorioPadre
|----manage.py
|----blog
        settings.py
        urls.py
        wsgi.py
        __init__.py
```

Figura 53: Estructura de un proyecto creado en Django.

- *manage.py* es un script que se encarga de la administración del sitio web que se va a crear. Este permite seleccionar y arrancar el servidor sobre el que va a ejecutarse la aplicación.

- *settings.py* es el scrip que contiene la configuración del sitio web. En este archivo se pueden configurar cosas como: la zona horaria, la codificación de la aplicación, las bases de datos que permiten almacenar datos en la aplicación, las rutas para los archivos estáticos, aspectos de seguridad y las aplicaciones instaladas, entre otras muchas cosas.

- *urls.py*. Contiene las declaraciones de las distintas URLs del proyecto, en forma de tabla de contenidos. Por defecto aparece definida la URL referente al sistema de administración del sitio web, como anteriormente comentábamos en las características del framework.

```
urlpatterns = [
    url(r'^admin/', admin.site.urls),
]
```

Figura 54: Fichero url.py con la URL correspondiente al administrador de la web.

- *__init.py__* es un fichero vacío encargado de la inicialización de los paquetes en Python.

- *wsgi.py* es un punto de entrada para que los servidores compatibles con la interfaz

WSGI permitan la conexión con nuestra aplicación web.

Para comprobar que el proyecto creado funciona es necesario iniciar el servidor web mediante el comando que aparece en la *Figura 55.*

```
C:\Users\ignac>manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).

You have 13 unapplied migration(s). Your project may not work properly until you apply the migrations
Run 'python manage.py migrate' to apply them.
June 14, 2017 - 11:26:48
Django version 1.11.2, using settings 'blog.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Figura 55: Comando para iniciar el servidor desde la cmd.

Se observa que aparece una alerta que indica que se deben aplicar migraciones en el proyecto. Las migraciones son como almacena Django los cambios realizados en el modelo del proyecto. Estas hay que aplicarlas cuando se crean nuevos archivos o cuando se producen cambios en el esquema de la base de datos.

```
C:\Users\ignac>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying sessions.0001_initial... OK
```

Figura 56: Migraciones aplicadas en el proyecto.

Tras aplicar las migraciones *(ver Figura 56)* se puede ver que al arrancar el servidor los warnings han desaparecido.

```
C:\Users\ignac>manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
June 14, 2017 - 11:35:35
Django version 1.11.2, using settings 'blog.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Figura 57: Correcta ejecución del servidor.

A continuación, ya es posible acceder al servidor, y a través del navegador se puede ver que el

proyecto ha sido creado con éxito (*ver Figura 58)*.

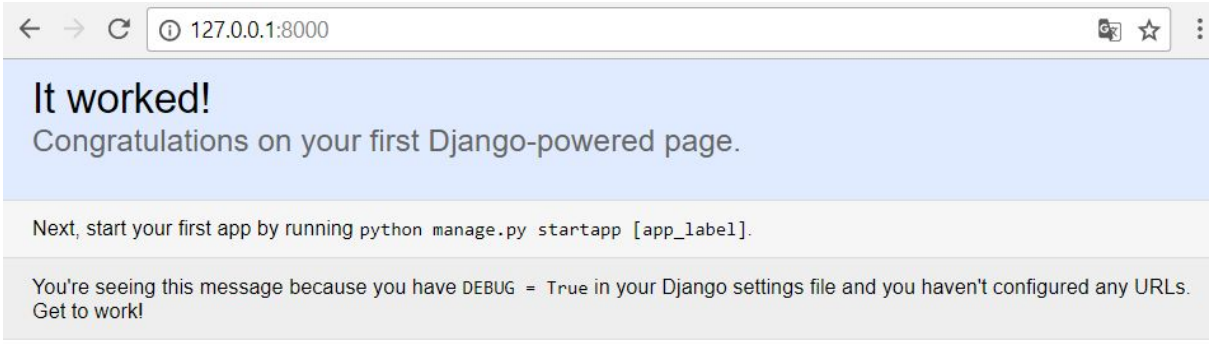


Figura 58: Mensaje que indica que el proyecto se ha creado correctamente.

PyCharm permite arrancar el servidor y consecuentemente, compilar el código de manera directa, haciendo click directamente en el triángulo verde del menú disponible.

Figura 59: Menú de PyCharm.

Configuración del directorio de trabajo

El último paso para poder empezar a trabajar, es definir el directorio de trabajo donde se va a desarrollar la aplicación web, para ello es necesario asegurarse que lo creamos dentro del directorio donde se encuentra el script *manage.py.*

```
C:\Users\ignac>manage.py startapp mvc
```

El directorio creado tiene la siguiente estructura:

```
mvc/
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
    migrations/
        __init__.py
```

Figura 60: Estructura del proyecto en Django.

Esta estructura está incompleta, para ello se ha necesitado crear un nuevo fichero *urls.py,* para definir las URLs dentro del directorio /mvc/~. A su vez, para definir el directorio /mvc/ como directorio raíz, es necesario modificar el fichero *urls.py*, definido dentro del del directorio /blog/, añadiendo el siguiente código:

```
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^mvc/', include('mvc.urls'))
]
```

Esto permitirá acceder al directorio /mvc/ y todos sus subdirectorios.

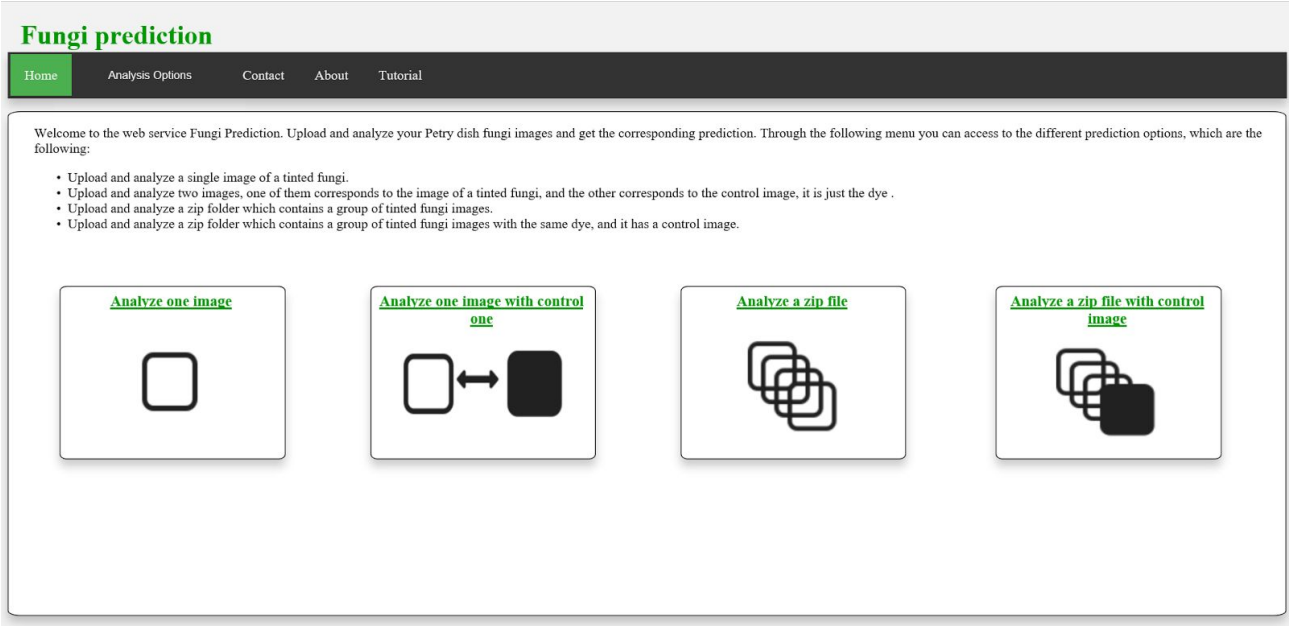
Para finalizar hay que indicar en el fichero de configuración *blog/settings.py,* el nombre de la aplicación correspondiente al directorio de trabajo que se ha creado. Para ello hay que añadir el siguiente código:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'mvc',
]
```

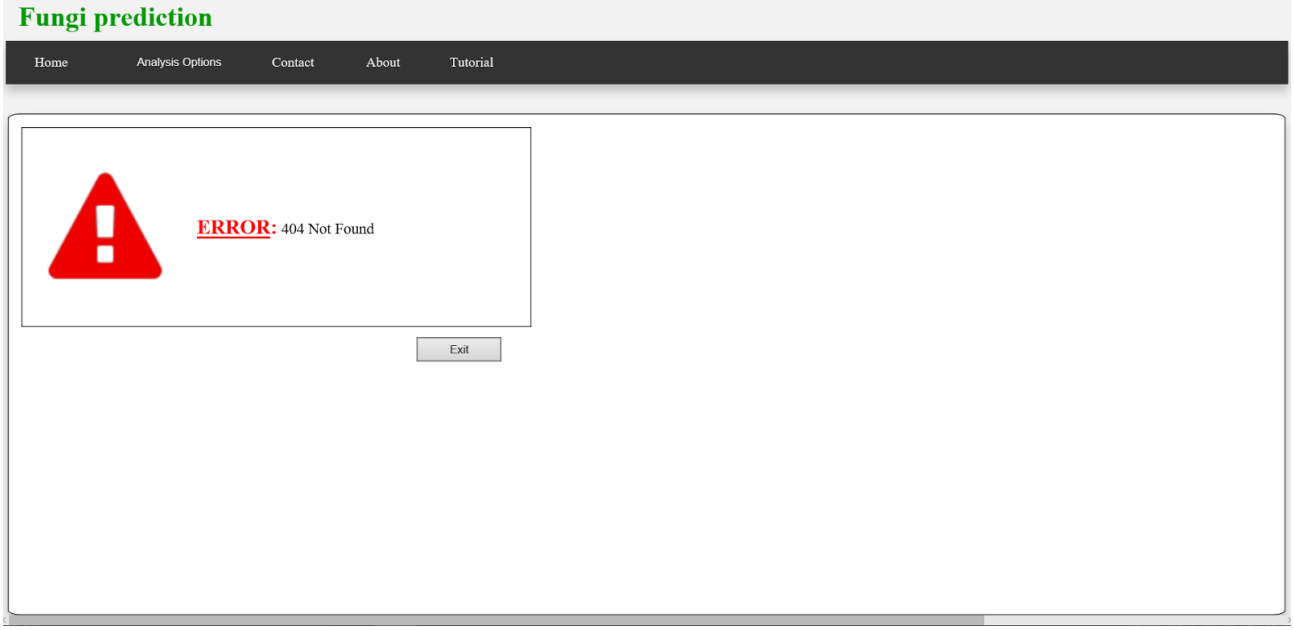
en la parte de aplicaciones instaladas.

B-Anexo II: Capturas finales de la aplicación

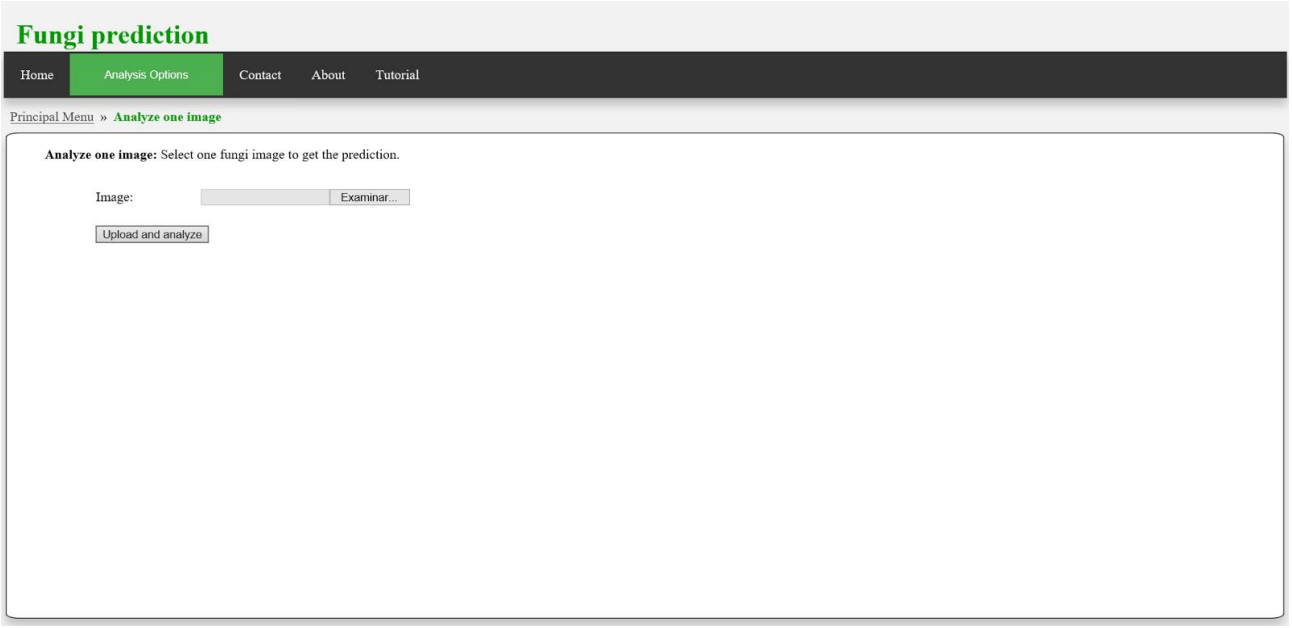
-Vista correspondiente al menú principal

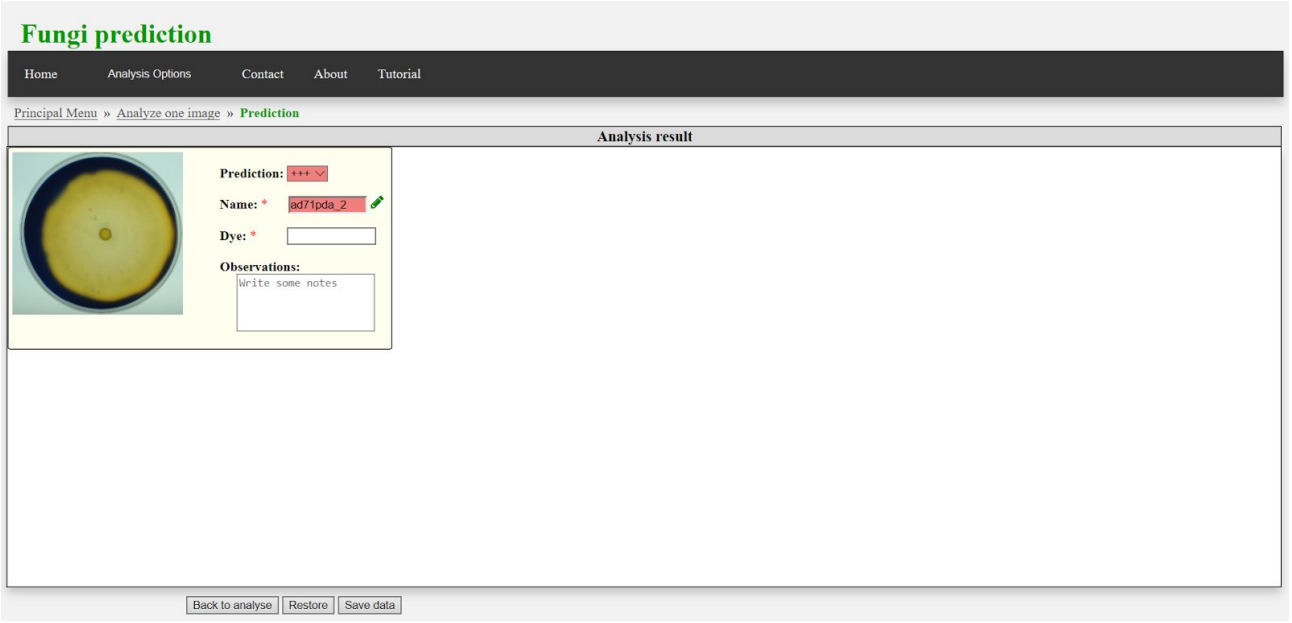


-Vista correspondiente a la página de error

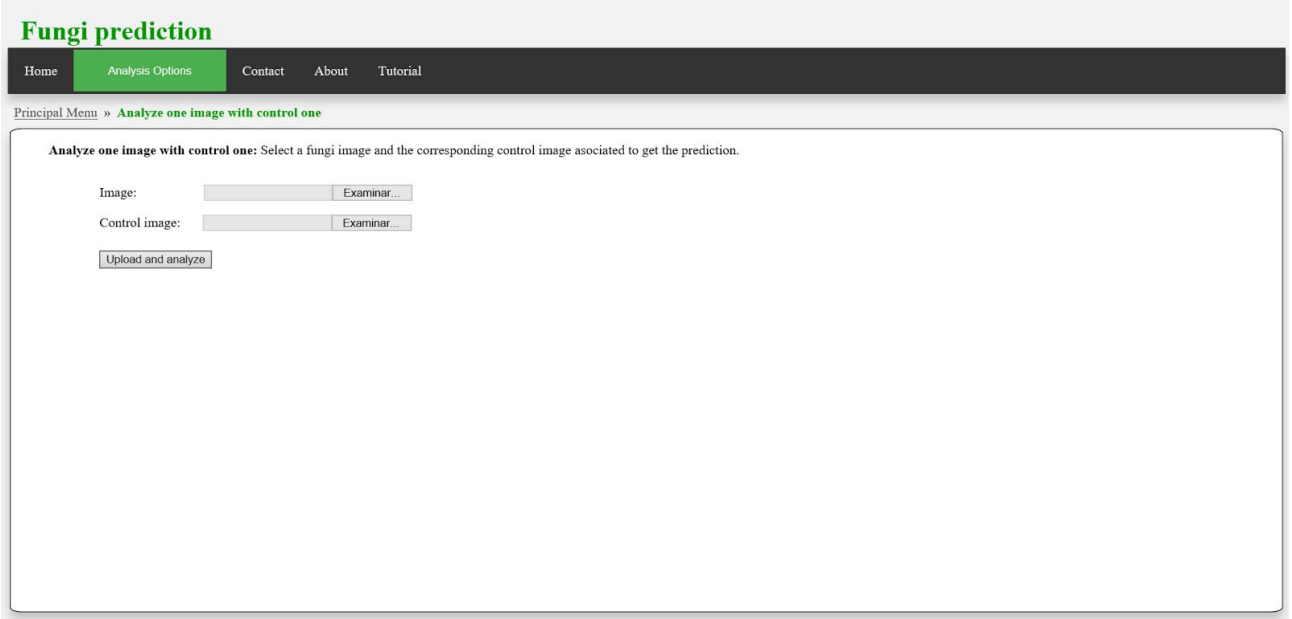


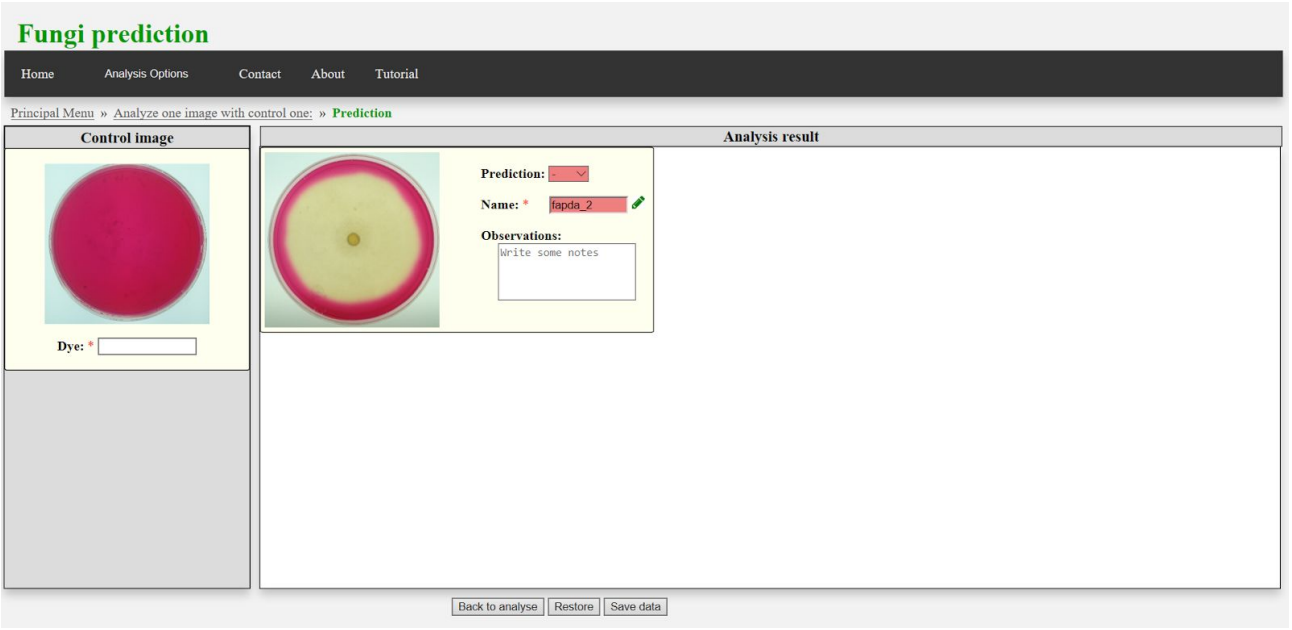
-Vistas correspondientes al método que permite analizar una única imagen.



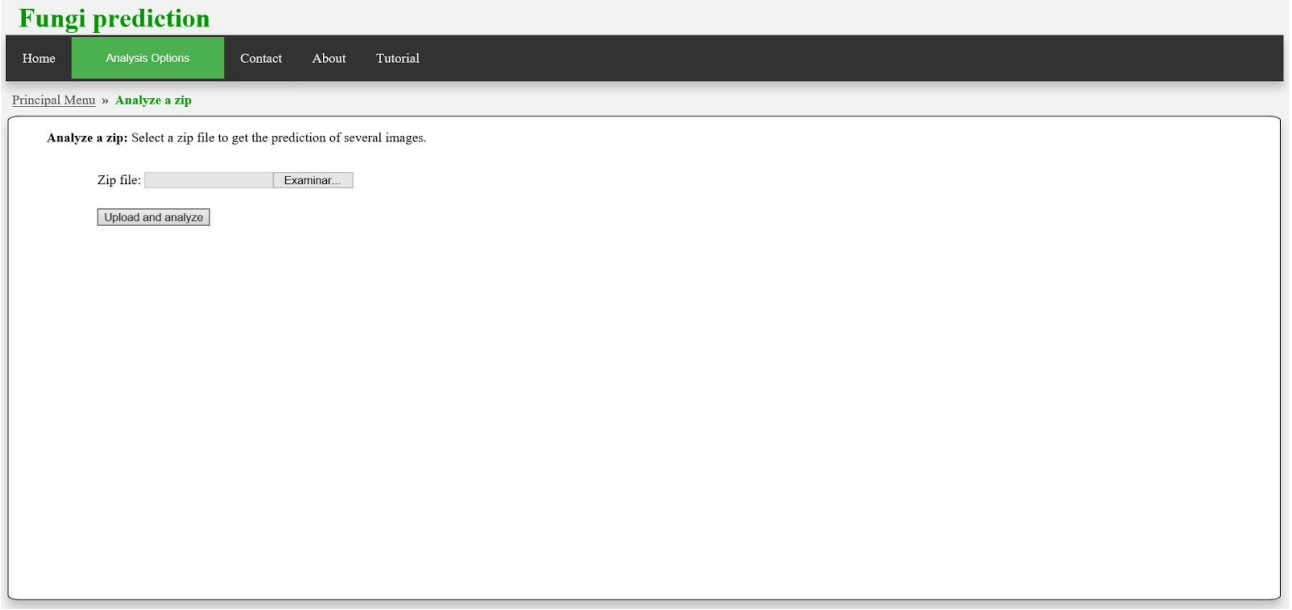


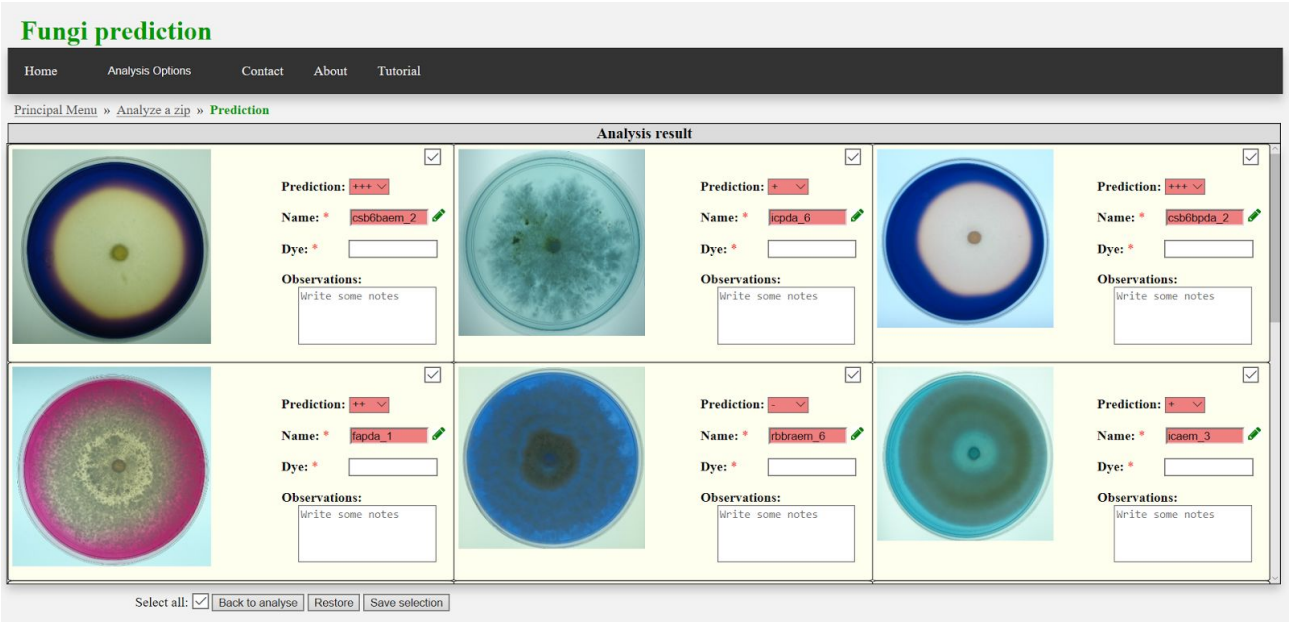
-Vistas correspondientes al método que permite analizar una imagen utilizando su correspondiente imagen de control asociada.



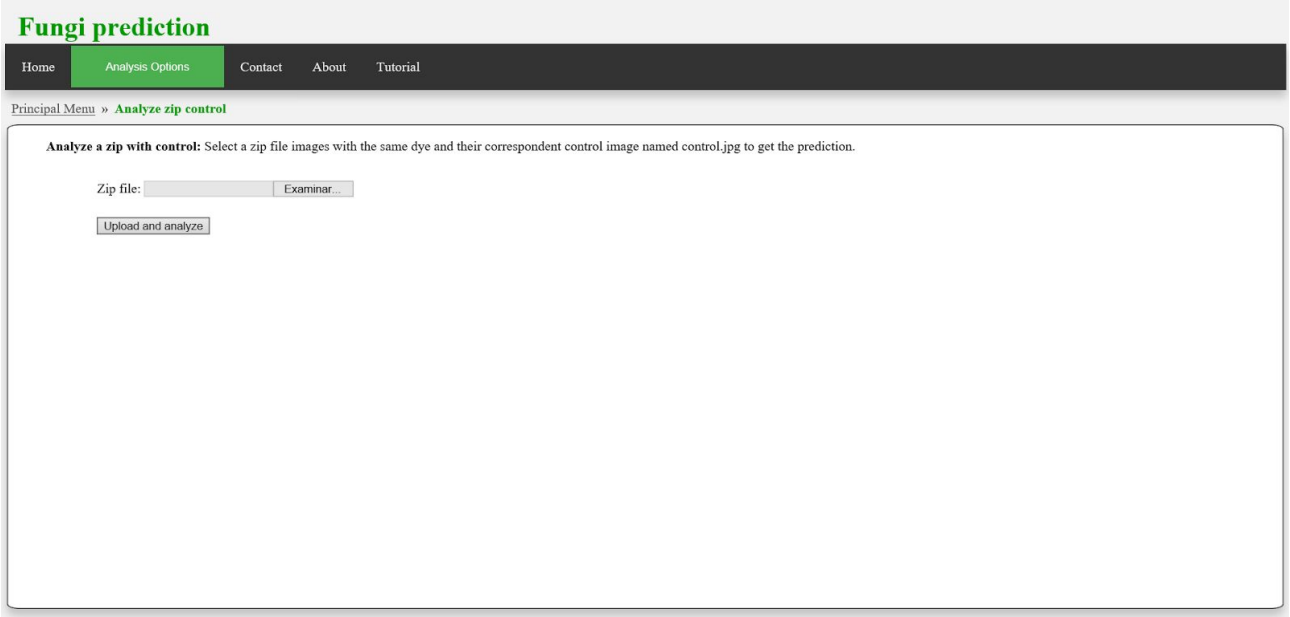


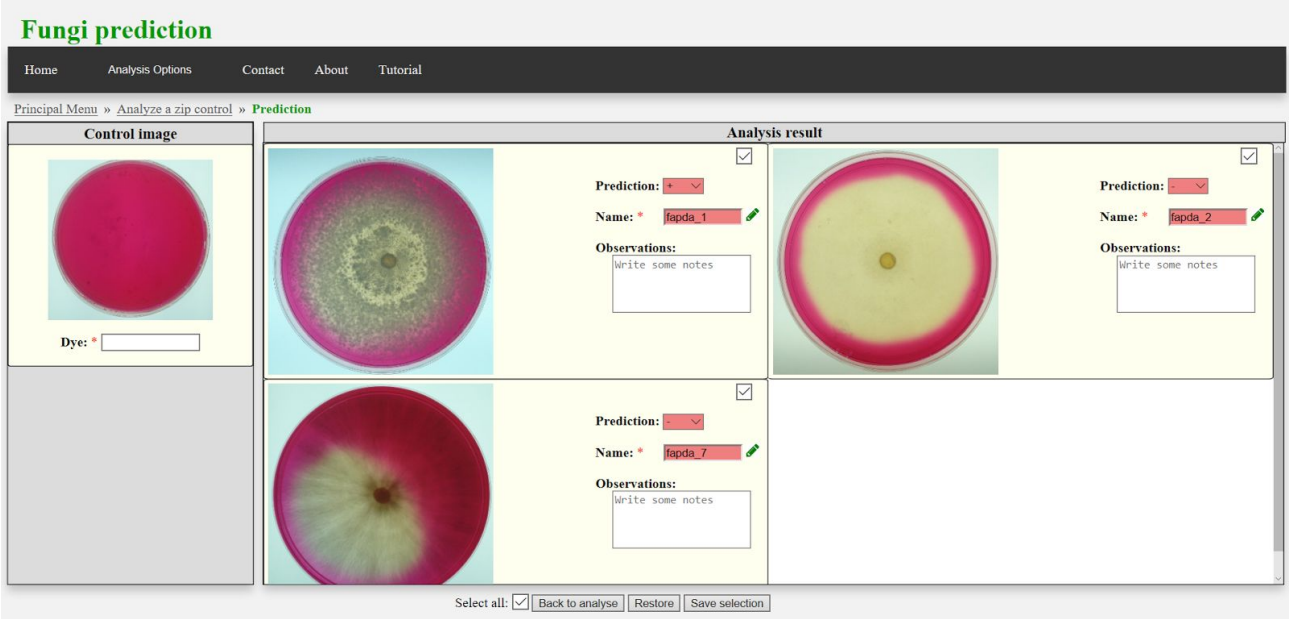
-Vistas correspondientes al método que permite analizar fichero zip de imágenes.





-Vistas correspondientes al método que permite analizar un fichero zip de imágenes con la imagen de control asociada.





-Tabla *mvc_ObjetoFungi* de la base de datos MySQL.

```
mysql> select* from mvc_objetofungi;
+----+-----------+------------+-----------------------------------------------------------+------------+-------------+
| ID | NAME      | DYE        | IMAGE                                                     | PREDICTION | DESCRIPTION |
+----+-----------+------------+-----------------------------------------------------------+------------+-------------+
|  1 | hongo1    | darkblue   | /media/fungi-classification/2017-07-20-14-27-43.jpg       | +++        | Good grow   |
|  2 | fungi2    | pink U45   | /mvc/media/control.jpg                                    | -          | Bad grow    |
|  3 | fungiZip1 | blue 43    | ../../../media/zip-files/2017-07-20-14-35-59/icaem_3.jpg  | +          |             |
|  4 | fungiZip2 | light blue | ../../../media/zip-files/2017-07-20-14-35-59/icaem_1.jpg  | ++         | TRANSPARENT |
|  5 | fapda_1   | Pink 456ZZ | /mvc/media/ZipControl/fapda_1.jpg                         | +          |             |
|  6 | fapda_2   | Pink 456ZZ | /mvc/media/ZipControl/fapda_2.jpg                         | +++        | The better  |
|  7 | fapda_7   | Pink 456ZZ | /mvc/media/ZipControl/fapda_7.jpg                         | +          |             |
+----+-----------+------------+-----------------------------------------------------------+------------+-------------+
7 rows in set (0.00 sec)
```